TAMPEREEN TEKNILLINEN YLIOPISTO

OLLI ROUHIAINEN

LOGIIKKAOHJATUN DATAN KERÄYKSEN INTEGROIMINEN PILVIPALVELUUN

Diplomityö

Tarkastaja: professori Seppo Kuikka
Tarkastaja ja aihe hyväksytty
Teknisten tieteiden
tiedekuntaneuvoston kokouksessa
4. marraskuuta 2015

TIIVISTELMÄ

Tampereen teknillinen yliopisto
Automaatiotekniikan koulutusohjelma
ROUHIAINEN, OLLI: Logiikkaohjatun datan keräyksen integroiminen pilvipalveluun
Diplomityö, 49 sivua, 1 liitesivu
Marraskuu 2015
Pääaine: Automaation ohjelmistotekniikka
Tarkastaja: professori Seppo Kuikka
Avainsanat: PLC, pilvipalvelu, automaatio, datan keräys, tietoturva

Teollisuudesta automaattisesti kerättävä tieto tarjoaa suuria mahdollisuuksia tuotantotehokkuuden parantamiseen. Saatavan tiedon määrä kasvaa hurjaa vauhtia ja järjestelmät integroituvat yhä tiiviimmin toisiinsa. Tuotantojärjestelmien älyn lisääntyminen tulee ohjaamaan teollisuuden toimintamalleja. Suuret tietomäärät eli niin sanottu big data tulevat tarjoamaan merkittäviä mahdollisuuksia prosessien kehittämiseen ja kokonaan uuden liiketoiminnan synnyttämiseen. Nykyaikaisissa automaatiojärjestelmissä ohjelmistojen merkitys korostuu hallittavien tietomäärien ja integraation kasvaessa. Erilaisilla suunnittelumalleilla pyritään ottamaan huomioon automaation erikoispiirteet ohjelmistosuunnittelussa. Myös yleiset ohjelmistotuotannon periaatteet, kuten käytettävyys ja ylläpidettävyys on huomioitava automaatiojärjestelmien ohjelmistoja kehitettäessä.

Tässä työssä käsitellään standardeja ja teknologioita, joiden avulla PLC:n suorittaman mittauksen tuloksia voidaan siirtää pilvipalveluun tarkasteltavaksi ja jalostettavaksi. Työssä käsitellään myös erilaisia tietoturvatekniikoita mahdollisesti käytettäväksi tiedonsiirrossa.

Osaa käsitellyistä tekniikoista on käytetty Inspector ohjelmistossa, jota on kehitetty ja testattu tämän työn käytännön osuudessa.

ABSTRACT

Tampere University of Technology
Master's Degree Programme Automation
ROUHIAINEN, OLLI: Integration of PLC based data collection to a cloud service
Master of Science Thesis, 49 pages, 1 Appendix page
November 2015
Major: Automation software engineering
Examiner: professor Seppo Kuikka
Keywords: PLC, cloud computing, automation, data collection, information security

Automatic data collection can help to improve efficiency of manufacturing industry. Amount of available data is increasing rapidly due to technology development. Integration of different systems to each other is nowadays a common and important development trend. Increasing intelligence of modern automation systems is changing traditional operations and improving processes. So called big data will also generate new business opportunities and completely new business models.

While amount of data is growing, importance of software in automation systems is growing too. Different design models and methods help to design good quality software and to handle special needs of automation environment. Properties as usability and maintainability are vital when designing software to complex and plausible safety related automation systems.

Different standards and technologies for transferring data from PLC to cloud service are studied in this thesis. Cloud computing will offer tools to visualize and process data for further use. Also different security methods are investigated to secure data transfer from PLC to servers.

Most suitable technologies are selected to be used in Inspector data collection product. Development of Inspector is done as a part of this thesis and testing was a part of the development process.

ALKUSANAT

Tämä diplomityö on tehty InSolution Oy:lle pitkällisen kyspyttelyvaiheen jälkeen pääosin syksyn 2015 aikana.

Haluan kiittää työn tarkastajaa professori Seppo Kuikkaa kärsivällisyydestä ja joustavasta yhteistyöstä. Lisäksi haluan osoittaa kiitokset myös työn tilaajana toimineelle ja toteutuksessa tukeneelle Juha Katajistolle tarjotusta mielenkiintoisesta tutkimusaiheesta, sekä muille InSolutionin työntekijöille työssä avustamisesta.

Kiitos myös vanhemmilleni tuesta sekä siskolleni avusta ja esimerkistä pitkän opintourani varrella.

Erityisen kiitoksen ansaitsee avopuolisoni työn loppukirin aikana kotiaskareiden laiminlyöntieni sietämisestä, allekirjoittaneen jatkuvasta kirjastoon hätistämisestä sekä tyttäremme kanssa piristävästä kannustamisestani.

Kiel, Saksa 21.11.2015

Olli Rouhiainen

SISÄLLYS

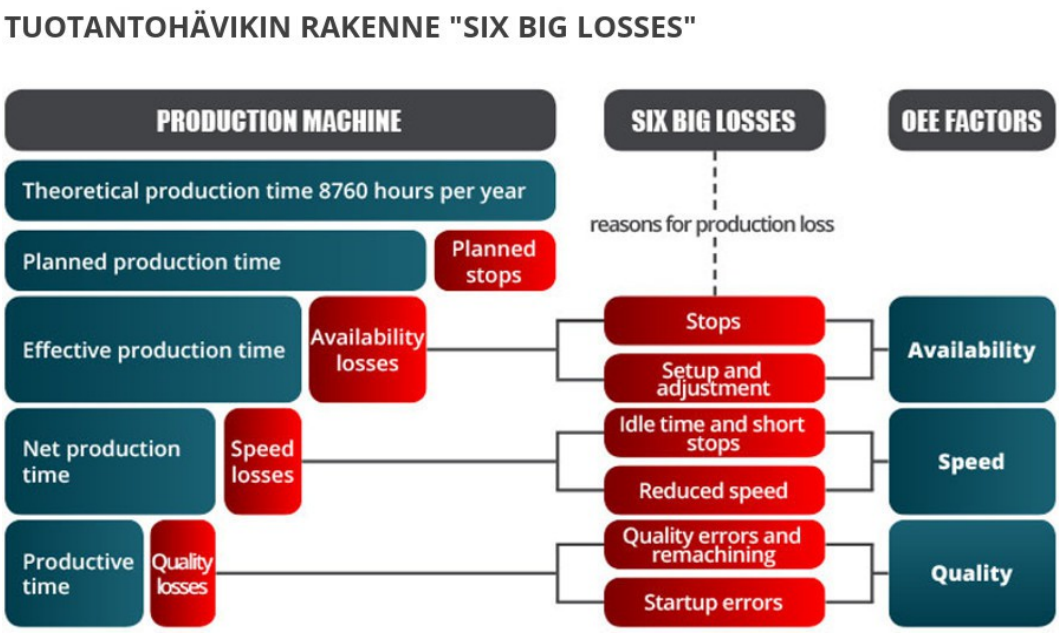
LYHENTEET, TERMIT JA NIIDEN MÄÄRITELMÄT

AES	Tiedon salausmenetelmä (Advanced Encryption Standard)
ERP	Tietojärjestelmä, jonka toiminnallisuus kattaa yrityksen toiminnan kaikki osa-alueet (Enterprise Resource Planning System).
EVO	Ohjelmistotuotannon iteratiivinen suunnitteluprosessi (Evolution Delivery)
HTTP	Asiakasohjelman palvelinkutsuihin käyttämä tiedonsiirtoprotokolla (Hypertext Transfer Protocol)
I/O	Ohjausjärjestelmän sisääntulot ja lähdot (Input/Output)
IoT	Esineiden internet (Internet of Things)
IPSec	Kokoelma tietoturvaprotokollia (IP Security Architecture)
JSON	Tiedonesitysmuoto (JavaScript Object Notation)
MES	Tuotannonohjausjärjestelmä (Manufacturing Execution System)
MVC	Tietosisällön, käyttöliittymän ja sovelluslogiikan erotteleva suunnittelumalli (Model – View – Control)
OEE	Tuotannon kokonaistehokkuutta kuvaava tunnusluku. (Overall Equipment Effectiveness)
OPC UA	Teollisuuden tarpeisiin kehitetty tiedonsiirtoprotokolla (OPC Unified Architecture)
PLC	Ohjelmoitava logiikka (Programmable Logic Controller)
RC4	Tiedon symmetrinen salausalgoritmi (Ron's Code 4 / Rivest Cipher 4)
ROPES	(Rapid Object-Oriented Process for Embedded Systems)
TCP/IP	Usean tietoliikenneverkkoprotokollan yhdistelmä (Transmission Control Protocol / Internet Protocol)
TLS	Verkkoliikenteen salausprotokolla (Transport Layer Security)
VPN	Tapa, jolla erilliset verkot voidaan yhdistää virtuaalisesti samaksi lähiverkoksi (Virtual Private Network)

1 JOHDANTO

Nykyään teollisuudessa pyritään tehostamaan prosesseja ja tuotantoa integroimalla alemman tason automaatiojärjestelmät ylemmän tason tietojärjestelmiin. Eri tuotantojärjestelmät voidaan yhdistää esimerkiksi tuotantolaitoksen laajuiseen MES-järjestelmään ja lopulta MES-järjestelmät voidaan liittää koko yrityksen laajuiseen ERP-järjestelmään. Integraatio mahdollistaa tuotannon reaaliaikaisen seurannan ja tuotannon optimoinnin.

Tuotantolaitteen kokonaistehokkuudella (Overall Equipment Efficiency, OEE) tarkoitetaan mittausmenetelmää, jolla pyritään määrittämään, kuinka tuottavasti tuotantolaite tai -linja suoriutuu sille asetetusta tehtävästä. OEE-laskenta perustuu kolmeen eri tekijään: käytettävyyteen, nopeuteen ja laatuun. Oleellinen osa tuotannon tehostamista onkin tunnistaa hävikin aiheuttavat syyt ja poistaa tuotantoa rajoittavat tekijät. Kuvassa 1 on havainnollistettu yleisimmät tuotantoa alentavat tekijät.



Kuva 1: Tuotantoajan hävikit (InSolution 2015)

Alkuun tietoa kerättiin manuaalisesti. Teknologian kehittyessä on alettu keräämään tietoa automaattisesti. Tästä seuraava suuntaus on kerätyn tiedon integrointi osaksi tuotannonohjausta ja yrityksen tietojärjestelmiä. Integraatioasteen kasvaessa myös datan analyyttinen tarkastelu ja tiedon jalostamisen merkitys korostuvat. Datamäärien kasvaessa on mahdollista tehdä entistä tarkempia ja monipuolisempia päätelmiä.

1.1 InSolution Oy

Tämän työn tilaaja InSolution Oy on teollisuutta palveleva kasvava PK-yritys. InSolutionin liiketoiminnot voidaan jakaa kolmeen osa-alueeseen, projektipalvelut, suunnittelu ja konsultointipalvelut sekä tiedonkeruu ja raportointituotteet. Erilaisina projektipalveluina teollisuudelle tarjotaan räätälöityjä automaatio- ja ohjelmistoratkaisuja tuotannon ja liiketoiminnan tarpeisiin. Oleellinen osa InSolutionin tarjoomaa on myös teollisuudelle tarjottavat suunnittelu- ja konsultointipalvelut. Nämä palvelut skaalautuvat asiakkaiden tarpeiden mukaan lyhytkestoisista konsultoinneista pitäaikaisiin ja jatkuviin yhteistyösopimuksiin. Tiedonkeruun ja raportoinnin tarpeisiin kehitetty Inspector-tuoteperhe tarjoaa työkalut erilaisten tuotannon ja laadun tunnuslukujen seurantaan ja raportointiin. Inspectoriin yhdistyvillä monipuolisilla ja räätälöitävillä liityntä- ja anturointiratkaisuilla tiedonkeruu on helppoa ja kustannustehokasta. (InSolution 2015)

1.2 Inspector

Inspector tiedonkeruujärjestelmällä on mahdollista saada reaaliaikaista tietoa tuotannosta havainnollisesti käyttäjälle esitettynä tai liitettynä muihin tietojärjestelmiin. Tuotannon seuranta mahdollistaa tuotantoa alentavien tekijöiden tunnistamisen ja tehokkuuden parantamisen näihin ongelmakohtiin puuttumalla.

Inspector-tuote on kehitetty jalostamalla ja jatkokehittämällä pitkään pääasiassa konepajateollisuutta palvellutta Fadector-ohjelmistoa. Fadector on alkujaan Fastems Oy:n kehittämä, pääasiassa metallintyöstökoneiden käyntitiedon seurantaan kehitetty tuote. Fadectoria on aikaisemmin käytetty lähinnä paikalliseen raportointiin, eikä siihen ole juurikaan toteutettu liityntöjä laajoihin käyntitieto- ja laadunseurantajärjestelmiin. Vuonna 2013 solmitulla liiketoimintakaupalla Fadector siirtyi InSolution Oy:lle, joka nykyään jatkokehittää ja ylläpitää ohjelmistoa asiakkainaan konepajateollisuuden lisäksi myös muu teollisuus ja eri toimialat. Näin jatkokehitetty ohjelmisto tunnetaan nykyään nimellä Inspector.

2 TYÖN TAVOITE JA AIHEEN RAJAUS

Tämä diplomityö käsittelee automaattisen datan keräyksen liittämistä pilvipalveluna toteutettuihin työkaluihin ja käyttöliittymään. Työ on osa InSolution Oy:n tiedonkeräystuotteen Inspectorin tuotekehitystä. Työllä tavoiteltuja hyötyjä ovat ymmärrys automaattisesti kerättävistä tiedoista ja niiden hyödyntämisestä, ohjelmiston PLC-osuuden kehittäminen, toimivuuden testaaminen sekä järjestelmän dokumentointi näiltä osin. Työn tuloksia tullaan hyödyntämään tiedonkeruu- ja raportointituotteessa, joka kerää määritettyjä tietoja mitattavasta järjestelmästä ja tallentaa tiedot pilvipalveluun, josta niitä voidaan esittää käyttäjälle tai siirtää eteenpäin muihin järjestelmiin.

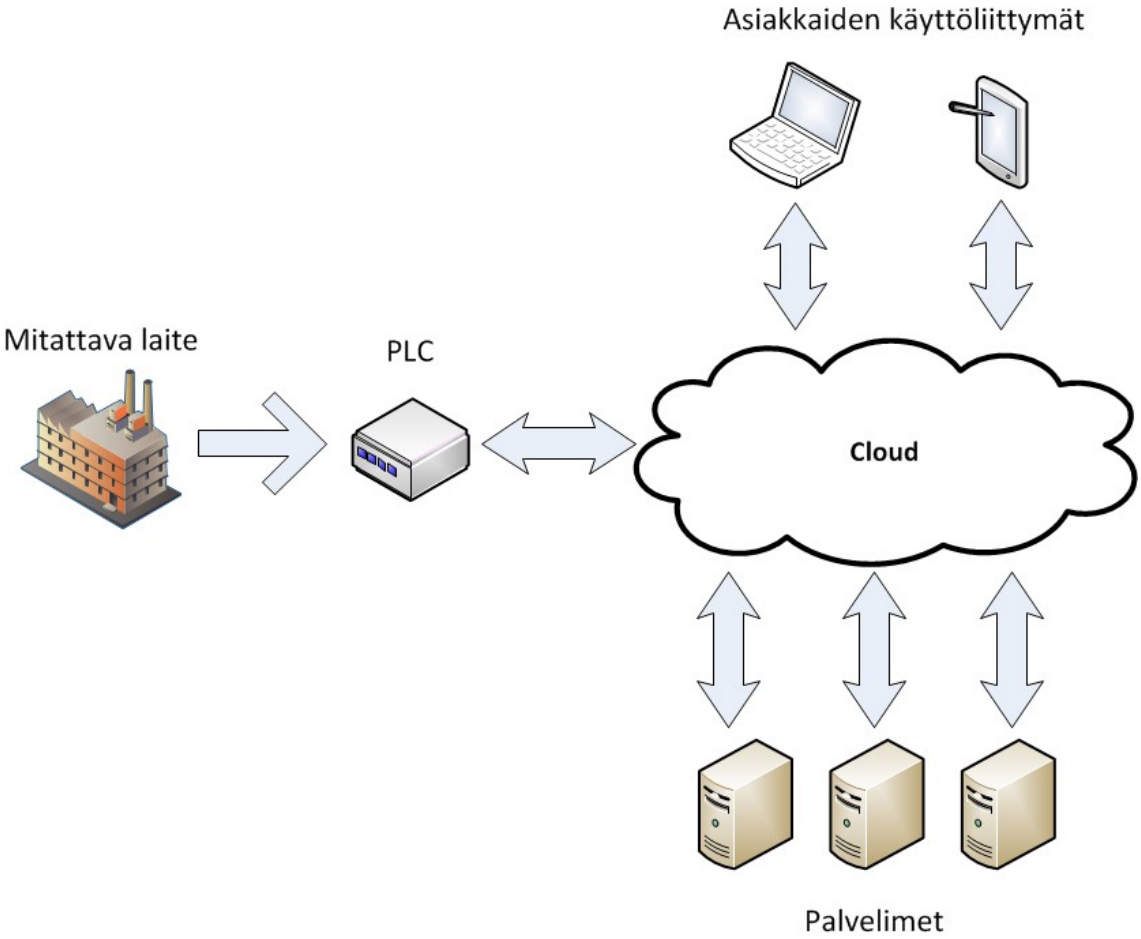
Tässä työssä keskitytään käsittelemään toteutettua PLC-osiota, sen liityntää mitattavaan laitteeseen tai prosessiin ja PLC:n keräämien tietojen välitystä palvelimelle. Työn alkuvaiheessa on tavoitteena rakentaa kuvan 2 mukaisella yleisarkkitehtuurilla toimiva tiedonkeruujärjestelmä, joka mahdollistaa kehitysvaiheessa kattavan testaamisen ja myöhemmin järjestelmän tuotantokäytön asiakkaalla. Tämän lisäksi tullaan toteuttamaan ohjelmoitavalle logiikalle automaattisia alustustoimintoja. Tavoitteena on tehdä logiikasta mahdollisimman modulaarinen ja automaattisesti konfiguroituva, jotta sen käyttöönotossa ei tarvittaisi lainkaan ohjelmisto-osaamista vaan pelkkä verkkoyhteyden ja käyttöjännitteen kytkeminen riittäisi. PLC:n on tällöin tarkoitus ottaa automaattisesti yhteys InSolutionin palvelimeen, jonka määrittelemänä se konfiguroituisi mahdollisimman pitkälle automaattisesti ja loppujen tarpeellisten asetuksien muuttaminen tapahtuisi verkkoyhteydellä ja erillisellä käyttöliittymällä.

Työn tavoitteisiin kuuluu selvitys teknologioista, joita käytetään tiedonsiirtoon Inspectorin PLC:n ja palvelimen välillä. Oleellinen osa työtä on myös tutkia erilaisia mekanismeja, joilla varmistetaan tiedon eheys ja toipuminen mahdollisissa ongelmatilanteissa, kun tietoja joudutaan puskuroimaan PLC:ssa. Työ pitää sisällään myös selvityksen kyseiseen sovellukseen soveltuvista tietoturvaratkaisuista PLC:n keräämien tietojen lähetyksen osalta. Työllä pyritäänkin saamaan vastaukset seuraaviin kysymyksiin:

- Mitä teknologioita tarvitaan ja voidaan soveltaa tiedon siirtoon PLC:n ja pilvipalvelun välillä?
- Miten tietoa voidaan puskuroida ongelmatilanteissa ja varmistua tiedon eheydestä?
- Mitä tietoturvaratkaisuja voidaan soveltaa verkossa siirrettävän tiedon salaamiseen?

Palvelimen ohjelmisto, jonka yksityiskohtainen tarkastelu jää tämän työn aihealueen ulkopuolelle, ylläpitää tietokantaa kerätystä informaatiosta. Palvelin myös jalostaa informaation asiakkaan käyttämää käyttöliittymää varten. Käyttöliittymän pysyessä

muusta ohjelmistosta erillisenä komponenttina, on mahdollista kehittää räätälöidyt käyttöliittymät eri käyttötarkoituksiin ja eri laitteille asiakkaan tarpeiden mukaan. Myöskään käyttöliittymiä ei käsitellä tässä työssä yksityiskohtaisesti näiden toteutuksien vaihdellessa web-selaimesta mobiililaitteisiin.



Kuva 2: Kehitettävän tiedonkeruujärjestelmän rakenne

Koska oleellinen osa ohjelmiston toteutusta on sen testaaminen, pitää PLC:n kehitys sisällään myös järjestelmän testaamista. Kehitysvaiheessa rakennetaan testiympäristö, joka käyttää todellista palvelinta. Aikomus on kuormittaa PLC:ta ja palvelinta, jotta havaitaan mahdolliset pullonkaulat. Tärkeä testattava asia on myös logiikkaohjaimen selviytyminen palvelimen ongelmista puskuroimalla dataa tarpeen mukaan eri ongelmatilanteissa. Tämä on tärkeä ominaisuus datan eheyden kannalta, sillä tietoa ei saa hävitä vaikka palvelimessa tai käyttöliittymässä esiintyisi ongelmia, joten työssä pyritään löytämään parhaat mekanismit datan puskurointiin ja datan eheyden varmistamiseen.

Työn teoriaosuudessa käsitellään myös yleisimmin mitattuja käyttövarmuuteen ja laatuun liittyviä tietoja ja niiden hyödyntämistä. Vaikka Inspectoria voidaan tulevaisuudessa soveltaa useilla eri toimialoilla ja erilaisten tietojen käsittelyssä,

paneudutaan tässä työssä tiedon keräämiseen ja analysointiin valmistavan teollisuuden tarpeiden mukaisesti.

3 TUTKIMUKSEN VIITEKEHYS JA TEOREETTINEN TAUSTA

3.1 Ohjelmoitavat logiikat

Ohjelmoitava logiikka eli PLC (Programmable logic controller) on pieni teollisuustietokone, jota käytetään prosessien ohjaukseen. Ohjelmoitava logiikka on liitetty järjestelmän instrumentointiin, eli sillä ohjataan järjestelmän toimilaitteita ja luetaan tietoja antureilta.

IEC (International Electrotechnical Commission) on määritellyt ohjelmoitavien logiikoiden ohjelmointikielet standardissaan IEC 61131-3. Nykyisin käytetyimpiä ohjelmointikieliä ovat IEC:n standardin mukaiset kielet. Standardi määrittelee viisi ohjelmointikieltä:

- IL (Instruction list) eli käskylista
- ST (Structured text) eli rakenteellinen teksti
- LD (Ladder logic) eli tikapuuohjelmointi
- FBD (Function block diagram) eli toimilohkokaavio
- SFC (Sequental function chart) eli vuokaavio

Standardissa määritellään mm. datatyypit, muuttujatyypit, ohjelmakomponentit, operaattorit sekä kunkin kielen toiminnallisuus.

Vaikka PLC-ohjelmointikielet on standardoitu ja logiikoiden toimintaperiaatteet ovat kaikilla valmistajilla samat, voi esimerkiksi fyysisten tulojen ja lähtöjen sitominen muistiosoitteisiin vaihdella jopa saman valmistajan eri tuotteiden välillä. Tästä syystä PLC-ohjelmat eivät yleensä ole täysin yhteensopivia, ja ohjelmat täytyykin aina testata lopulta käytettävällä laitteistolla.

3.1.1 PLC:n toimintaperiaate

PLC:n ohjelmoinnissa on otettava huomioon PLC:n toimintaperiaate, erityisesti ohjelman suorituksen syklisyys. Ohjelman toimintaperiaate voidaankin jakaa karkeasti kolmeen vaiheeseen. Ensimmäisessä vaiheessa luetaan kaikkien PLC:hen liitettyjen tulojen arvot, mikä tarkoittaa esimerkiksi antureiden tilojen lukemista. Toisessa vaiheessa suoritetaan koko ohjelmakoodi kokonaisuudessaan, minkä tuloksena lähdöille lasketaan uudet arvot koodissa määritetyllä tavalla. Kolmannessa vaiheessa lähtöjen arvot ohjataan fyysisiin lähtöihin eli toimilaitteiden ohjauksiksi. Nämä kaikki kolme vaihetta on suoritettava aina määrätyssä syklin ajassa, joka on yleensä suuruusluokaltaan 1-10ms, riippuen kuitenkin paljon sovelluksesta. Jos kaikkia vaiheita ei ehditä suorittaa syklin aikana, on kontrollerin toiminta määrittelemätöntä ja johtaa ohjelman toimimattomuuteen. Sykliä toistetaan niin pitkään kun ohjelmoitava logiikka on käynnissä.

Ohjelman komponentit voidaan toteuttaa kolmena eri tyyppinä. Perinteinen aliohjelma on tyypiltään ”Function”. Sille voidaan antaa parametreja ja sen paluuarvo voi olla tyypiltään jokin perustietotyypeistä tai ohjelmoijan itse määrittelemistä tietotyypeistä. Toinen ohjelmakomponenttityyppi on ”Program”. Se voi parametrien (VAR_INPUT) lisäksi sisältää useita ohjelmoijan määrittelemiä lähtöjä (VAR_OUTPUT). Lisäksi sille voidaan antaa viiteparametreja (VAR_IN_OUT), eli se voi käyttää ja muokata suoraan kutsujan määrittelemää muuttujaa. Lisäksi sillä voi olla omia sisäisiä muuttujia (VAR) ja se muistaa oman tilansa PLC-syklien välillä. Kolmas ja eniten käytetty tyyppi on toimilohko (englanniksi function block). Se vastaa ominaisuuksiltaan program-tyyppiä, mutta merkittävänä erona näillä on komponenttien instanssien käsittely. Program-tyypistä ei voi luoda useita instansseja ja näin ollen niitä ei tarvitse esitellä muuttujalistauksissa. Ne ovat siis globaaleja yksittäisiä toimilohkoja. Toimilohkon instanssi puolestaan määritellään ja nimetään aina, se tietää oman tilansa ja sen näkyvyys vastaa olio-ohjelmoinnista tuttuja periaatteita. Funtion blockeja voidaankin käyttää ikään kuin olioita, jolloin PLC-ohjelmoinnissa on mahdollista käyttää olio-ohjelmoinnin hyödyllisiä periaatteita, vaikka PLC:lla ei kaikkia olio-ohjelmoinnin mekanismeja, kuten periyttämistä ja kuormittamista, ole mahdollista helposti toteuttaakaan.

3.1.2 Kenttäväylät

Kenttäväylät ovat teollisuustietoverkkojen standardeja ja protokollia, joita käytetään reaaliaikaiseen ja hajautettuun ohjaukseen. PLC voidaan liittää erilaisiin kenttäväyliin, mikä mahdollistaa instrumentoinnin hajauttamisen. Nykyään teollisuudessa useimmin käytettyjä kenttäväylätekniikkoja ovat mm. AS-i (Actuator Sensor Interface), Ethercat, Profinet, Profibus, Interbus ja Modbus sekä erityisesti ajoneuvoissa käytetty CAN-väylä (Controller Area Network). Ethernetiin perustuvat kenttäväylät, kuten Ethercat, ovat selvästi yleistymässä, mutta automaatiojärjestelmien elinkaarien ollessa pitkiä, ovat vanhatkin kenttäväylät edelleen varsin yleisessä käytössä. Kenttäväyliä on määritelty standardissa IEC 61158.

3.1.3 Tulot ja lähdöt

Ohjelmoitavan logiikan ulkoisista liitännöistä käytetään yleisesti termejä tulo ja lähtö. Nimitykset juontuvat termeistä input/output, joista käytetään usein lyhennettä I/O. Tuloporttien kautta logiikka saa tietoa järjestelmän tilasta, ja lähtöporttien kautta se ohjaa järjestelmää. I/O:n kautta PLC-ohjelma liitetään ohjattavaan fyysiseen prosessiin.

Digitaaliset signaalit ilmaisevat päällä tai poissa tilan (0 tai 1). Digitaalisten signaalien siirtoon käytetään yleensä jännitettä tai virtaa. Tällöin tietty jännitteen tai virran arvo tulkitaan 0-tilaksi ja toinen 1-tilaksi. Useimmiten nykyaikaiset ohjelmoitavat logiikat käyttävät 24 V:n DC-jännitettä, jolloin riittävän korkea jännite tulkitaan arvoksi tosi ja matala jännite arvoksi epätosi.

Analogiset signaalit ilmaisevat tietyn arvon määritellyllä vaihteluvälillä. Analoginen signaali muunnetaan PLC:ssa numeeriseksi arvoksi, jonka tarkkuus riippuu käytettävästä tietotyypistä. Analogisia signaaleja käytetään yleensä mitattaessa haluttua suuretta, esimerkiksi lämpötilaa, painetta tai painoa. Useimmiten käytetyt analogiset signaalit ovat välillä 0-10V jänniteviestillä ja välillä 4-20mA käytettäessä virtaviestiä.

Esimerkki analogisesta signaalista löytyy tämän työn tuloksia hyödyntävästä asiakasprojektista. Toteutettavassa ratkaisussa tullaan mittaamaan värähtelyanturilla teollisuuspuhaltimen värähtelyä, josta voidaan päätellä, onko puhallin käynnissä ja voidaan tunnistaa mahdollinen vikaantuminen tai muu niin sanottu ryntääminen. Käytettävä anturi välittää mittaamansa värähtelyn voimakkuuden tehollisarvon (true RMS) PLC:lle 4-20mA virtaviestinä, jonka tehtävä on skaalata viesti halutulle asteikolle.

3.2 Tiedonsiirron protokollat

Tässä kappaleessa esitellään tärkeimmät protokollat, joita käytetään PLC:n keräämän tiedon siirtämisessä Inspector palveluun.

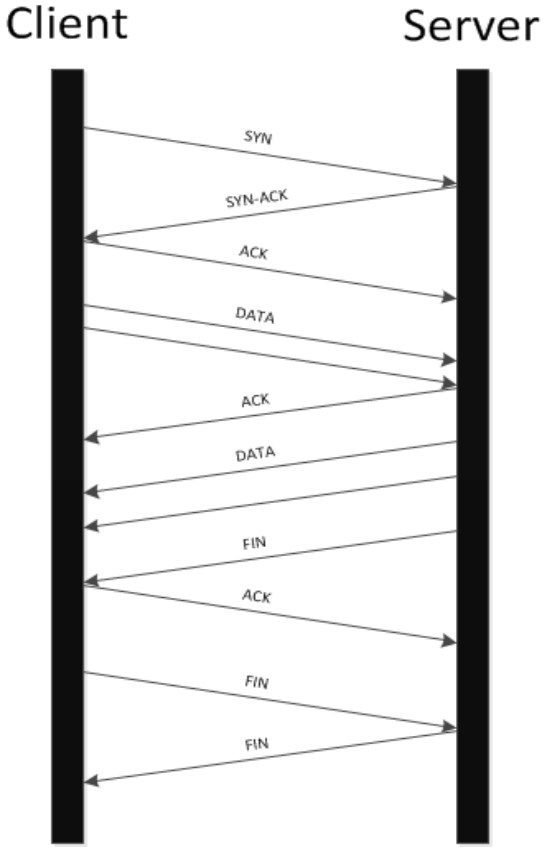
3.2.1 TCP/IP

TCP/IP-protokollaperhe (Transmission Control Protocol / Internet Protocol) koostuu useasta tietoverkkojen käyttämästä protokollasta ja sillä on merkittävä rooli useimpien internetiä käyttävien palveluiden mahdollistajana. TCP/IP-protokollien standardoinnista vastaa IETF-standardointiorganisaatio (The Internet Engineering Task Force), joka on suuri kansainvälinen avoin yhteisö. IP-protokolla vastaa laitteiden osoitteistamisesta ja pakettien reitittämisestä verkossa. Sen päällä voidaan ajaa useita muita verkkoprotokollia, joista TCP-protokolla on yleisimmin käytetty.

IP-Protokolla
IP-Protokolla toimii parhaan suorituskyvyn periaatteella, eli se ei pidä sisällään pakettien uudelleen lähettämistä. Kyseessä on yhteydetön tiedonsiirtotapa, eli IP ei ylläpidä erikseen yhteyttä pakettien siirtoa varten, vaan jokainen paketti pitää pystyä reitittämään ja käsittelemään irrallaan muista paketeista jopa satunnaisessa järjestyksessä. (Fall & Stevens 2012)

TCP-Protokolla
TCP-protokolla huolehtii kahden laitteen välisestä tiedonsiirrosta muodostamalla ja järjestämällä siirrettävät paketit ja lähettämällä mahdollisesti hukkuneet paketit uudestaan. TCP-yhteys pitää sisällään kolme vaihetta: yhteyden luominen, tiedon siirto ja yhteyden katkaisu. Yhteys muodostetaan kolmitiekättelyllä. Kolmitiekättelyssä yhteyden aloittaja lähettää ensiksi kohdelaitteelle SYN-paketin. Vastaanottajan saatua

SYN-paketin, vastaa tämä aloittajalle SYN/ACK-paketilla merkkinä SYN-paketin saapumisesta. Lopuksi yhteyden aloittaja vastaa kohteelle ACK-paketilla merkiksi, että on ottanut SYN/ACK-paketin vastaan. Yhteyden ollessa auki tietoa voidaan siirtää datapaketteina molempiin suuntiin. Suljettaessa yhteys molempien osapuolien täytyy ilmoittaa lopettavansa yhteys erikseen lähettämällä FIN-paketti, jonka vastaanottaja kuittaa vastaanottamakseen ACK-paketilla. Vaikka toisen suuntainen yhteys olisi jo katkaistu, on toiseen suuntaan yhä yhteys auki, kunnes sekin suljetaan. TCP-yhteyden viestiliikenne on esitetty kuvassa 3. (Comer 2000)



Kuva 3: TCP-viestiliikenne

3.2.2 JSON

JSON on lyhenne englanninkielisestä nimestään JavaScript Object Notation. JSON on yksinkertainen tiedon esitysmuoto, joka nimestään huolimatta ei ole riippuvainen javascriptistä. JSON:lle on ominaista mm. helppo ymmärrettävyys ihmiselle luettavassa muodossa, sekä yksinkertainen automaattinen käsiteltävyys. Koska tieto on helposti luettavassa muodossa, on myös tietoturva otettava huomioon sovelluskehityksessä. Inspectorin tietoturvaa käsitellään tarkemmin kappaleessa 3.7. JSON rakentuu kahdesta yksinkertaisesta rakennemuodosta, muuttujan nimen ja arvon muodostamista pareista sekä määritetyn järjestyksen omaavasta listasta. Näille rakenteille löytyy vastaavuudet

käytännössä kaikista ohjelmointikielistä. Esimerkiksi ohjelmoitavien logiikoiden ST-kielellä nimi-arvo-parit muodostavat struct-muuttujia ja listarakennetta vastaa array-tietotyyppi. JSON-rakenteessa alkioiden arvot voivat olla tyypiltään joko olio, lista, numero, merkkijono, ”true”, ”false” tai ”null” (JSON 2015).

Alla esimerkki yksinkertaisesta JSON rakenteesta:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

3.2.3 HTTP

HTTP-lyhenne tulee englanninkielisistä sanoista Hypertext Transfer Protocol. Protokollan toiminta perustuu asiakasohjelman lähettämiin pyyntöihin, joihin palvelu lähettää vastauksen. Pyynnöt ja vastaukset siirretään yleensä käyttämällä TCP-yhteyttä.

Ensimmäinen versio HTTP:sta tuki vain GET-metodia. Sittemmin standardia on laajennettu kattamaan monipuolisempia tarpeita. Viimeisin versio on vuonna 2015 julkaistu HTTP/2. HTTP/2 päivitys ei tuonut muutoksia metodien määrittelyyn, vaan ne ovat täysin yhteensopivia edellisen HTTP version 1.1 kanssa, joka on tällä hetkellä varsin yleisesti käytetty versio (RFC7540 2015).

Alla on listattu HTTP-protokollan määrittelemät metodit uusimpien HTTP-versioiden mukaan:

- GET, pyytää palvelimelta halutun resurssin
- POST, lähettää tietoja palveluun
- HEAD, pyytää palvelimelta vain halutun resurssin otsikkotiedot, joista voidaan päätellä koko resurssia lataamatta, onko resurssi muuttunut
- OPTIONS, kysyy palvelimen tai tietyn resurssin ominaisuuksia
- PUT, pyytää resurssin tallentamista määriteltyyn osoitteeseen
- DELETE, pyytää tietyn resurssin poistamista
- TRACE, pyytää palvelua kaiuttamaan vastaanottamansa pyynnön sellaisenaan takaisin asiakasohjelmalle, mikä helpottaa vianetsintää
- CONNECT, avaa yhteyden, joka jää päälle

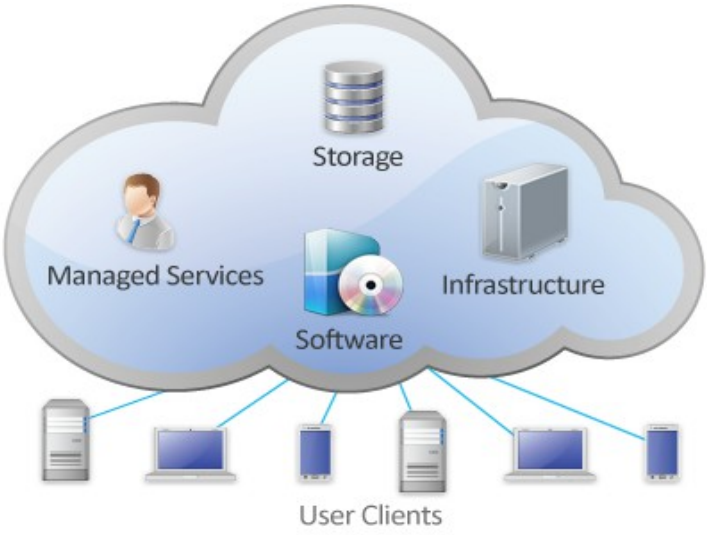
Palvelin vastaa pyyntöihin vastauskoodilla, joka kertoo asiakasohjelmalle pyynnön tuloksen. Vastauskoodit muodostuvat kolmesta numerosta ja selitteestä. Numerolla 2

alkavat vastauskoodit tarkoittavat pyynnön onnistunutta toteuttamista. Numerolla 3 alkavat vastaukset kertovat pyynnön kohteena olevan resurssin löytyvän jostain muusta osoitteesta. Numerolla 4 alkavat vastauskoodit merkitsevät pyynnössä olevaa asiakasohjelman tekemää virhettä ja numerolla 5 alkavat koodit tarkoittavat palvelimesta johtuvaa virhettä pyynnön suorituksessa. (RFC7231 2014)

3.3 Pilvipalvelut

Pilvipalveluilla eli pilvilaskennalla (Cloud computing) tarkoitetaan tietoteknisten palveluiden seuraavassa kuvattavaa hajautusta tietoverkkoon. Pilvipalvelun välityksellä jaettuja resursseja ja informaatiota voidaan jakaa tarpeen mukaan eri käyttäjille. Pilvipalveluiden tarjoamien rajapintojen johdosta loppukäyttäjää ei yleensä pakoteta käyttämään tiettyä käyttöjärjestelmää tai esimerkiksi tiettyä selainta, vaan sovellukset ovat riippumattomia käytettävästä alustasta. Palvelun käyttämä verkko voi olla tyypiltään joko julkinen pilvi, yksityinen pilvi tai näiden yhdistelmä. Pilven tyyppi määritellään sen mukaan, kenellä on pääsy kyseiseen pilveen ja kenen omistuksessa se on. Julkisen pilven infrastruktuuri sijaitsee jaetuilla palvelimilla, joita käyttävät samanaikaisesti muutkin samaa pilvipalveluntarjoajaa käyttävät tahot. Yksityisen pilven infrastruktuuri on varattu ainoastaan kyseisen pilven käyttöön ja se muodostaa suljetun verkon.

Koska pilvipalvelun loppukäyttäjä näkee vain käyttämänsä palvelun käyttöliittymän, ei käyttäjä huomaa välttämättä eroa hajautetun pilvipalvelun tai paikallisen sovelluksen välillä. Ideana on siis piilottaa palvelun ja palvelimien toteutusyksityiskohdat ja mahdollinen hajautus asiakassovelluksilta, jolloin erilaisia toteutuksia voidaan kuvata yksinkertaisesti pilvenä, kuten kuvassa 4.



Kuva 4: Pilvipalvelun esitys

Yksinkertaisimmillaan pilvilaskentaa käytetään laskennan hajauttamisessa, tilapäisten kuormien tasaamisessa, ohjelmiston osien tai kokonaisuuksien ulkoistamisessa ja vastaavissa pelkkää laskentatehoa kaipaavissa toiminnoissa. Pilvilaskenta muuttaa yleistyessään myös ohjelmistotuotannon liiketoimintamalleja. Pilvipalveluiden hinnoittelumallit perustuvat usein käytön mukaiseen jatkuvaan laskutukseen, eikä perinteiseen kiinteään lisenssimaksuun.

Yleisimmät pilvipalveluiden palvelutyypit ovat SaaS (Software as a Service), IaaS (Infrastructure as a Service) ja PaaS (Platform as a Service). SaaS tarkoittaa ohjelmiston hankkimista palveluna, jolloin ohjelmistosta maksetaan käytön mukaan, toisin kuin perinteisessä lisenssipohjaisessa toimintamallissa. Useimmiten SaaS-ohjelmistojen palveluita käytetään selainkäyttöliittymällä, jollainen on myös Inspectorille jo toteutettu. IaaS tarkoittaa palvelininfrastruktuurin ulkoistamista, eli palveluntarjoaja tarjoaa tallennustilaa ja laskentatehoa asiakkailleen. PaaS tarkoittaa koko palvelualustan ulkoistamista. Tämän alustan päälle asiakkaat voivat kehittää sovelluksiaan. (NIST 2011)

3.4 Esineiden internet

Esineiden internetillä (Internet of Things, IoT) tarkoitetaan yleensä erilaisten laitteiden ja esineiden kykyä yhdistyä internetiin tai toisiinsa, mikä mahdollistaa kyseisten laitteiden tilojen seurannan ja ohjauksen verkon yli. Laitteiden määrän kasvaessa myös kerättävissä olevan tiedon määrä kasvaa räjähdysmäisesti. Tästä suuresta määrästä kerättävää raakadataa käytetään englanninkielistä nimitystä ”Big data”. (Evans et al. 2012) Teollinen internet sisältää luonnostaan paljon liityntöjä tuotantolaitteisiin ja koneisiin, joten käsitteinä esineiden internetillä on paljon yhteistä teollisen internetin kanssa. Raakadatan kerääminen ja etenkin jalostaminen luo huikeita liiketoimintamahdollisuuksia informaation merkityksen jatkuvasti kasvaessa myös teollisuudessa.

Internetin laajentuminen myös laitteisiin on tuonut mukanaan erityisesti Saksassa mainetta saavuttaneen käsitteen Industry 4.0, jolla kuvataan tapahtumassa olevaa neljättä teollista vallankumousta. Ensimmäinen vallankumous oli mekaanisten laitteiden, kuten höyrykoneen ja kehruukoneen, tuoma muutos tavaroiden valmistuksessa. Toinen vallankumous oli sähkökäyttöisen massatuotannon yleistyminen, josta tunnetuin esimerkki on Fordin T-mallin valmistus sarjatuotantona. Kolmannen vallankumouksen katsotaan alkaneen 1970-luvulla elektroniikan ja informaatioteknologian yleistyessä. Nyt tapahtuva neljäs vallankumous muodostuu esineiden, laitteiden ja erilaisten järjestelmien verkottumisella niiden muodostaessa niin sanottuja kyberfyysisiä järjestelmiä (engl. CPS, Cyber Physical System). Merkittävä kehitystä edistävä tekijä on IPv6, joka mahdollistaa yksiselitteisen osoitteen määrittelemisen kaikille verkkoon liittyville laitteille. (Kagermann et al. 2013)

Tuotantojärjestelmien älykkyyden lisääntyminen ja järjestelmiin liittyvän informaation määrän kasvu ja käytön tehostuminen muokkaa teollisuuden toimintatapoja kautta linjan. Tietoa kulkee koko ajan entistä enemmän niin horisontaalisesti, eli samalla tasolla olevista tuotantojärjestelmistä ja laitteista toisiin, kuin myös vertikaalisesti, eli esimerkiksi tuotantolaitteilta MES-järjestelmään ja edelleen ERP-järjestelmään ja takaisin. Tällä integroitumisella tullaan saavuttamaan huomattavia hyötyjä tuotannon joustavuudelle ja jäljitettävyydelle.

3.4.1 MTConnect

MTConnect on avoin standardi, joka määrittelee tiedon siirtoa tuotantolaitteilta erilaisille sovelluksille, kuten MES- ja ERP-järjestelmille. Tiedon siirto on määritelty xml-muotoon ja sen välitys tapahtuu HTTP-protokollan (Hypertext Transfer Protocol) mukaisesti käyttäen valittua verkkoprotokollaa, useimmiten Ethernet-protokollaa. MTConnect määrittelee tiedon lukemistavan laitteelta muiden järjestelmien käyttöön, mutta se ei käsittele tiedon, esimerkiksi ohjauspyyntöjen, siirtoa laitteelle. (MTConnect 2014)

Myös Inspector voidaan tulevaisuudessa valjastaa käyttämään hyväkseen suoraan laitteiden tarjoamaa MTConnect rajapintaa, koska Inspectorin arkkitehtuuri mahdollistaa myös muiden sovelluksien, kuin oman PLC:n, liittämisen kyseiseen pilvipalveluun. Jos MTConnectin käyttö yleistyy työstökoneissa, tulee PLC:n suorittaman tilanseurannan tarve vähenemään ja työstökoneen seurannan liittäminen Inspectoriin helpottumaan. Vaikka MTConnect vaikuttaa saavan tulevaisuudessa vahvan jalansijan työstökoneiden parissa, tulee myös PLC:n suorittamalle mittaukselle kuitenkin olemaan jatkossakin tarvetta etenkin muilla teollisuuden aloilla. Lisäksi metalliteollisuudessa konekanta uudistuu hitaasti, joten myös PLC:n suorittamalle seurannalle on varmasti myös jatkossa tarvetta työstökoneidenkin parissa.

3.4.2 UPC UA

Laitteiden väliseen tiedonsiirtoon on kehitetty standardi OPC Unified Architecture, jota ylläpitää OPC foundation. Standardia kehitettäessä tärkeänä periaatteena on ollut palvelulähtöinen arkkitehtuuri (Service Oriented Architecture, SOA). OPC:n (OLE for Process Control) aikaisempi versio, OPC Classic, on laajasti käytetty, mutta on riippuvainen Microsoftin COM (Component Object Model) ja DCOM (Distributed COM) teknologioista. OPC Classic toimii siis vain Windows ympäristössä, kun taas OPC UA on nimensä mukaisesti alustariippumaton. Alkuperäisen OPC:n tietoturvaominaisuudet ovat varsin puutteelliset, mikä lisäsi myös tarvetta uudistetulle standardille. OPC UA:ssa tiedonsiirtoon voidaan käyttää yleisiä tiedonsiirtoprotokollia, kuten TCP ja HTTP, mikä omalta osaltaan edesauttaa standardin laajennettavuutta ja alustariippumattomuutta. OPC UA palvelimelle ja asiakasohjelmille löytyy valmiita toteutuksia vapaasti käytettäväksi esimerkiksi C#, Java ja C-kielillä toteutettuina.

OPC UA standardi koostuu seuraavista osista:

1. Overview and Concepts
2. Security Model
3. Address Space Model
4. Services
5. Information Model
6. Mappings
7. Profiles
8. Data Access
9. Alarms and Conditions
10. Programs
11. Historical Access
12. Discovery
13. Aggregates

(OPC Foundation 2015)

SOA-periaatteiden mukaisesti toiminta perustuu siihen, että OPC UA-palvelin, esimerkiksi PLC, tarjoaa informaatiota, jonka asiakasohjelma pystyy lukemaan dynaamisesti. Spesifikaatio määrittelee kaksi tapaa tiedon esitykseen, binääri- ja XML-muodon. Binäärimuotoinen esitystapa on tehokas rajallisten resurssien ympäristöissä, kuten sulautetuissa järjestelmissä, kun taas XML-muoto tarjoaa paljon valmiita työkaluja monipuoliseen tiedon käsittelyyn. (Mahnke et al. 20009)

Tietoturva on ollut vahvasti mukana suunniteltaessa OPC UA:ta ja se sisältääkin valmiita ratkaisuja tietoturvasta huolehtimiseen. Standardi määrittelee kaksi eri tietoturvaprotokollaa, WS-SecureConversation ja UA-SecureConverstion. Näitä OPC UA:n tarjoamia tietoturvaratkaisuja käsitellään tarkemmin kappaleessa 3.7.

OPC UA määrittelee myös kaksi tiedonsiirtoprotokollaa, UA TCP ja SOAP/HTTP -protokollat. Protokollien avulla huolehditaan yhteyden muodostuksesta ja hallinnasta. UA TCP on yksinkertainen protokolla, joka sisältää kuitenkin mekanismin, jolla istunto voi toipua automaattisesti yhteyskatkoista. SOAP/HTTP on lyhenne englanninkielisistä protokollista Simple Obejct Access Protocol ja Hyper Text Transfer Protocol. Näillä protokollilla voidaan toteuttaa www-sovelluspalvelu (engl. Web service). SOAP käyttää XML-formaattia tiedon esitykseen. HTTP:n sijaan voitaisiin tiedonsiirtoon käyttää myös salauksen sisältävää HTTPS-protokollaa, mutta käytettäessä OPC UA:n tarjoamaa salausmekanismia tämä ei ole tarpeen. (Mahnke et al. 20009)

3.5 Tiedon keräys ja käsittely

Nykyaikaiset digitaaliset teknologiat mahdollistavat suurien tietomäärien keräämisen automatisoidusti. Kerättyä tietoa jalostamalla voidaan saavuttaa merkittävää hyötyä

liiketoiminnalle ja toisaalta kehittää tiedon jalostamisesta kokonaan uutta liiketoimintaa kuten Inspector omalta osaltaan osoittaa.

Tiedonkeruun tärkeimpiä vaatimuksia on oikeellisuus, nopeus ja kustannustehokkuus. Näitä kaikkia ominaisuuksia pystytään useimmiten parantamaan tiedonkeruun automatisoinnilla. (Leclin 2006)

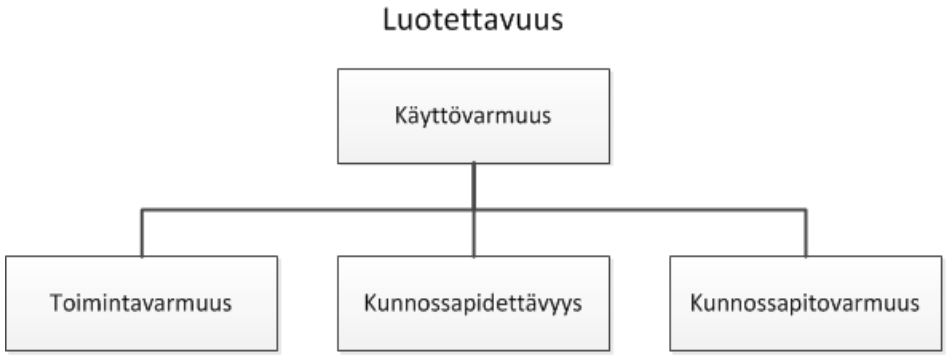
3.5.1 Tiedon luokittelu

Tietoa voidaan luokitella ja määritellä useilla eri tavoilla. Eräs luokittelu on jakaa tieto kahteen lajiin, hiljaiseen ja eksplisiittiseen tietoon. Hiljaisen tiedon osuus kaikesta tiedosta on huomattavan suuri. Hiljainen tieto, englanniksi tacit knowledge, on pääasiassa sosiaalisessa vuorovaikutuksessa syntyvää ja kokemusperäistä yksilöihin sitoutunutta tietoa. Se ilmenee yleensä intuitiona, arvoina sekä uskomuksina ja sen ilmaiseminen sekä tallentaminen on vaikeaa. Sitä on myös vaikea suojata ja pitää organisaation sisällä. Eksplisiittinen tieto, englanniksi explicit knowledge, on tarkasti määriteltyä ja muodoltaan numeerista tai muuten yksiselitteistä ja tarkasti määriteltävissä. Sen käsitteleminen ja tallentaminen on helppoa ja sitä voidaan yhdistellä muun olemassa olevan eksplisiittisen tiedon kanssa. Eksplisiittistä tietoa on myös helppo siirtää ja sitä esiintyy eri tavoilla tallennettuna, esimerkiksi taulukoissa, kirjoissa ja tietokannoissa. Eksplisiittinen tieto soveltuukin hyvin automaattisesti kerättäväksi. (Sydänmaanlakka 2007)

Tietoa voidaan luokitella myös jakamalla se eri tasoihin tiedon merkityksen perusteella. Yleensä tämä luokittelu tehdään jakamalla tiedot kolmeen perustasoon, data, informaatio ja tietämys. Data on yksinkertaisinta mahdollista tietoa, jota voidaan jalostaa informaatioksi. Data tarkoittaa jalostamattomia, yksittäisiä tiedon osia. Se on pääosin numeerista eikä sillä ole yksinään merkitystä. Data muuttuu hyödylliseksi jalostettaessa se informaatioksi. Informaatio on siis käsiteltyä dataa, jolle on muodostunut jokin tietty merkitys. Tietämys on näistä tiedon perustasoista jalostetuinta, mikä tarkoittaa entisestään jalostettua informaatiota, joka pitää sisällään kokemuksen, oppimisen, ajattelun ja sisäistämisen tuomaa näkemystä. (Laihonen et al. 2013)

3.5.2 Käyttövarmuus

Puhuttaessa järjestelmän tai laitteen luotettavuudesta, tarkoitetaan yleensä kyseisen laitteen käyttövarmuutta, joka koostuu IEC 60050(191) -standardin mukaisesti kolmesta osatekijästä: toimintavarmuudesta, kunnossapidettävyydestä ja kunnossapitovarmuudesta kuvan 5 mukaisesti.



Kuva 5: Käyttövarmuus IEC 60050(191) standardin mukaan

Käyttövarmuutta kuvaavia tunnuslukuja on määritelty IEC 60050(191) -standardissa. Tärkeimpiä käyttövarmuuden mittareita ovat toimintakelpoisuusaika (UT, Up Time) ja toimintakelvottomuusaika (DT, Down Time). Yleisin käyttövarmuuden mittari on käytettävyys (Availability). Käytettävyys ilmoitetaan tavallisesti prosenttilukuna ja se tarkoittaa toimintakelpoisuusajan suhdetta tarveaikaan.

Toimintavarmuus kuvaa koneen toimivuutta ilman häiriöitä, kunhan ennalta määritellyt huollot toteutetaan määritellysti. Kunnossapidettävyys kertoo, kuinka helppoa ja nopeaa huoltaminen ja korjaaminen ovat. Kunnossapitovarmuus puolestaan kuvaa toimintaa valvovan ja ylläpitävän organisaation tehokkuutta toimintakyvyn ylläpitämiseksi.

Käyttövarmuuden osatekijät määritellään Jänneksen mukaan standardeissa SFS-IEC 50(191) ja SFS-EN 60300-1 seuraavasti.

- **Toimintavarmuus**: Kohteen kyky suorittaa vaadittu toiminto määrätyissä olosuhteissa vaaditun ajanjakson. Toimintavarmuutta voidaan kuvata esimerkiksi TTF-luvulla (time to failure), joka tarkoittaa aikaa, joka kuluu käyttöönotosta kohteen ensimmäiseen vikaantumiseen
- **Kunnossapidettävyys**: Kohteen kyky olla pidettävissä tilassa tai palautettavissa tilaan, jossa se pystyy suorittamaan vaaditun toiminnon määritellyissä olosuhteissa käyttäen vaadittuja menetelmiä ja resursseja. Esimerkiksi TTR (time to restoration) tarkoittaa vian jälkeiseen kunnostukseen kuluvaa aikaa
- **Kunnossapitovarmuus**: Kuvaa toimintaa ylläpitävän organisaation kykyä suorittaa vaadittu tehtävä tehokkaasti määrätyissä olosuhteissa vaaditulla ajanhetkellä. Kunnossapitovarmuuden mittareina käytetään mm. korjauksen odotusaikaa (WT, waiting time), hallinnollista viivettä (AD, administrative delay) ja logistista viivettä (LD, logistic delay). Näillä ilmaistaan kunnossapito-organisaation kykyä reagoida vikaantumiseen.

(Jännes 2011)

Usein edellä mainittujen tunnuslukujen lyhenteiden aluissa käytetään kirjainta M (Mean), joka tarkoittaa keskimääräistä tunnusluvun arvoa. Tunnusluvut eivät ole yksiselitteisesti standardoituja, joten tunnuslukuja käytettäessä on hyvä esittää myös sen laskentakaava. (Välisalo & Rouhiainen 2000)

3.5.3 Laadun mittaus

Laadun mittaamisessa on aina otettava huomioon, mitä laadulla tarkoitetaan kyseisessä tapauksessa. Laadulla voidaan tarkoittaa esimerkiksi asiakkaan kokemaa lisäarvoa, teknisten vaatimusten täyttämistä tai ominaisuuksien vaihteluiden hallintaa. Yleensä laatu voidaan määritellä myös soveltuvuudeksi käyttötarkoitustaan varten. Juran käyttää laadun käsitteelle yleistynyttä määritelmää ”Fitneess for use”, joka korostaa asiakkaiden ja heidän tarpeidensa todellista ymmärrystä (Juran 1998). Laadun systemaattinen seuranta ja kehittäminen edellyttää laadun mittaamista määritellyssä viitekehyksessä. Tätä varten on kehitetty useita eri tunnuslukuja eri käyttötapauksia varten.

3.5.4 Tehokkuuden mittaus

Tuotannon kokonaistehokkuutta mitataan yleensä KNL-laskennalla (Käytettävyys, nopeus, laatu) josta käytetään usein englanninkielistä nimitystä OEE (Overall Equipment Effectiveness). OEE-arvo lasketaan kertomalla käytettävyys, nopeus ja laatu keskenään. Tätä arvoa mittaamalla ja seuraamalla on helppo havainnoida tuotannon tehokkuudessa tapahtuneet muutokset. Yksinkertainen tapa parantaa tuotannon tehokkuutta on tunnistaa tehokkuutta pienentävät tekijät, joihin on tarpeen puuttua. Nämä tuotantoa haittaavat tekijät voidaan jakaa käytettävyyshävikkiin (suunnitellut pysäytykset ja laiterikot), nopeushävikkiin (katkokset tuotannossa ja hidastunut nopeus) ja laatuhävikkiin (hylätyt tuotteet ja käynnistyksessä syntynyt hävikki). OEE-luvun laskenta on määritelty standardissa PSK 7501 - Prosessiteollisuuden kunnossapidon tunnusluvut.

3.6 Automaatiojärjestelmien ohjelmistokehitys

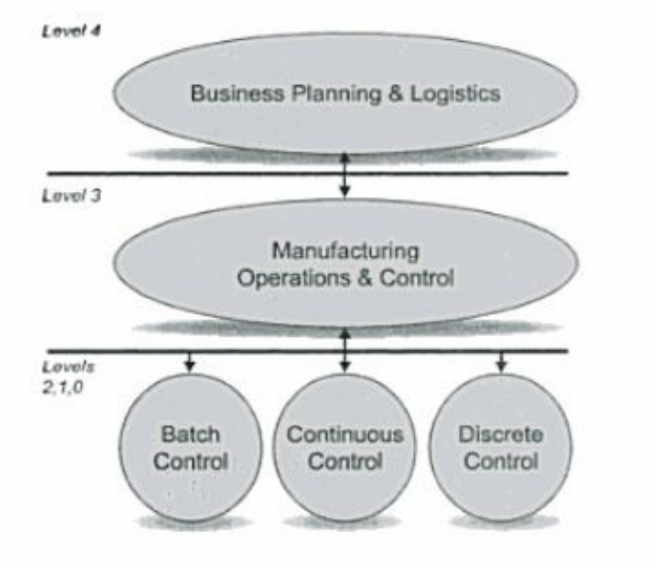
Ohjelmistojen merkitys automaatiojärjestelmissä kasvaa jatkuvasti. Teollisuuden käyttämien automaatiojärjestelmien ohjelmistojen vaatimukset poikkeavat esimerkiksi perinteisistä toimistosovelluksista. Niiden virheellinen toiminta voi aiheuttaa laitevaurioita, suuriakin taloudellisia tappioita tai jopa henkilöturvallisuuden vaarantumisen. Automaatioa varten kehitetyissä ohjelmistoissa niin sanotut RAMS-tekijät, eli luotettavuus (Reliability), toiminnallinen käytettävyys (Availability), ylläpidettävyys (Maintainability) ja turvallisuus (Safety) ovat erityisen tärkeitä. Automaation ohjelmistojen kehityksessä korostuu myös kaikelle ohjelmistokehitykselle yhteisten vaatimusten lisäksi ymmärrys fyysisistä prosesseista, koska automaation

ohjelmistoa kehitetään usein rinnakkain laitesuunnittelun kanssa, osaksi prosessia ohjaavaa automaatiojärjestelmää. (Kuikka 2007)

Tässä kappaleessa esitellään ohjelmistosuunnittelun malleja, jotka soveltuvat hyvin erityisesti automaatiojärjestelmien suunnitteluun ja kehitykseen.

3.6.1 Automaatiojärjestelmän hierarkiamalli

Teollisuusjärjestelmien jakautumista eri abstraktiotasoihin mallinnetaan usein hierarkiamallilla. ISA-95 on standardi rajapintojen ja integraation kehitykseen yrityksen tietojärjestelmien ja ohjausjärjestelmien välillä. Kuvassa 6 on esitetty ISA-95:n mukainen tuotantojärjestelmän hierarkiamalli, joka jakaa yrityksen toiminnot hierarkiatasoihin nollasta neljään.



Kuva 6: Kuva 3: Tuotantojärjestelmän hierarkiamalli (ISA-95.00.01-2000, lähteen Scholten 2007 mukaan)

Taso neljä kattaa koko yrityksen laajuisen liiketoiminnan hallinnan. Tämä taso pitää sisällään yrityksen toimintojen pidemmän aikavälin suunnittelun, kuten materiaalihankinnan, tuotantoaikataulun ja logistiikan suunnittelun sekä tilausten hallinnan. Tason neljä järjestelmiä voidaan kutsua yleisesti toiminnanohjausjärjestelmiksi tai ERP-järjestelmiksi (Enterprise Resourse Planning). Näiden järjestelmien avulla tuotetaan tarvittavia tietoja, jotka välitetään alemman tason järjestelmille ja näin vaikutetaan alemmalla tasolla toimivaan valmistuksen ohjaukseen.

Taso kolme käsittää valmistuksen ohjauksen toiminnot. Se pitää sisällään yksityiskohtaisen tuotannon hallinnan, esimerkiksi tuotannon optimoinnin ja raportoinnin laitostasolla. Tämän tason järjestelmiä kutsutaan usein MES-järjestelmiksi

(Manufacturing Execution System). Tämän tason oleellinen tehtävä on myös toimia rajapintana tason neljä järjestelmien ja perusautomaation välillä, koostamalla ja välittämällä reaaliaikaista tietoa tuotannon tapahtumista ylemmälle tasolle. Samalle se myös sopeuttaa tuotantoa ylemmältä tasolta tulevan ohjauksen perusteella.

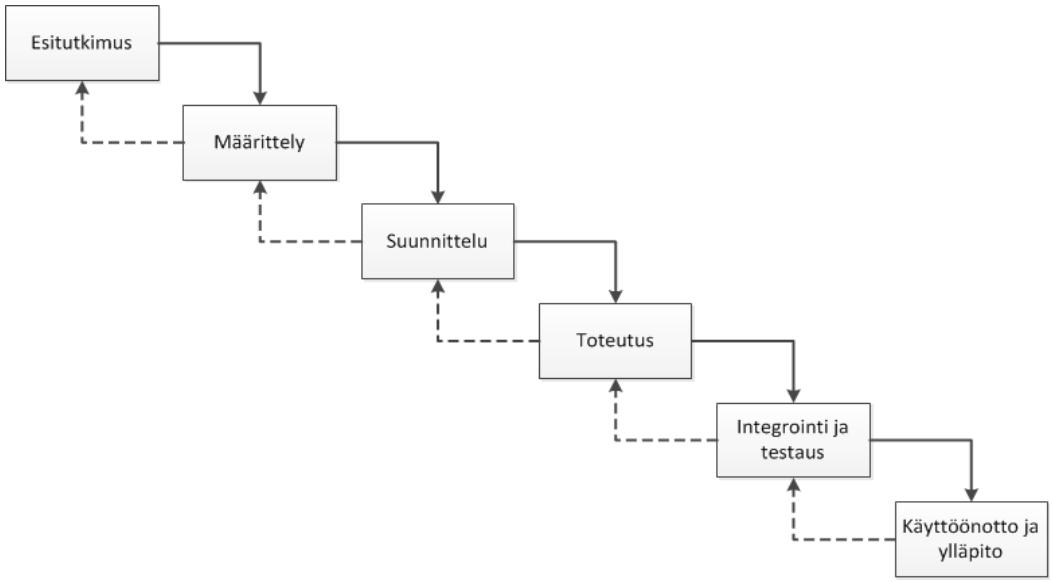
Tasot nolla, yksi ja kaksi kattavat perusautomaatiojärjestelmän, eli järjestelmän laitetason ohjauksen sekä säädön, mittaukset ja tarpeelliset suojaukset. Tasolla kaksi tarkoitetaan automatisoitua prosessin ohjausta, laitteiden ohjausjärjestelmiä. Taso yksi kattaa järjestelmän instrumentoinnin, eli anturoinnin ja erilaiset ohjattavat toimilaitteet. Taso nolla tarkoittaa varsinaista fyysistä prosessia, joka ei kuulu enää varsinaiseen automaatiojärjestelmän toimintaan.

3.6.2 Vesiputousmalli

Vesiputousmalli on klassinen ohjelmistotuotannon malli, jossa kehitysprosessi etenee vaiheittain. Vesiputousmallia seurattaessa edetään järjestyksessä yhdestä vaiheesta toiseen, edellisen vaiheen valmistuttua siirrytään seuraavaan vaiheeseen eikä ideaalitapauksessa edelliseen vaiheeseen enää palata. Vesiputousmalli korostaa varhaisen suunnittelun merkitystä, aikaisemmissa vaiheissa tehdyt panostukset helpottavat myöhempiä vaiheita. Vesiputousmalli on selkeä ja helposti ymmärrettävä malli, joka ohjaa panostamaan kattavaan dokumentaatioon. Myös projektin eteneminen on helppo hahmottaa suoritettavan vaiheen perusteella.

Vesiputousmalli soveltuu useimmiten vain pienille kokonaisuuksille, joiden toiminta ja vaatimukset pystytään määrittelemään riittävän tarkasti jo kehityksen alkuvaiheessa. Käytännön ohjelmistosuunnittelussa kuitenkin tulee usein muutostarpeita ja uusia vaatimuksia, joiden hallintaan vesiputousmalli soveltuu huonosti.

Vesiputousmallin puutteista huolimatta se soveltuu hyvin laitekehityksen kanssa rinnakkain tehtävään ohjelmistokehitykseen. Laitesuunnittelun alkuvaiheessa tehtyjen ratkaisujen merkitys on huomattavan suuri ja myöhemmin muutoksien tekeminen on työlästä, joten huolellisesti tehdyn esitutkimuksen ja määrittelyn merkitys korostuu. Esimerkiksi turva-automaatiosovelluksiin vesiputousmalli soveltuu hyvin kehitysprosessin selkeyden vuoksi.



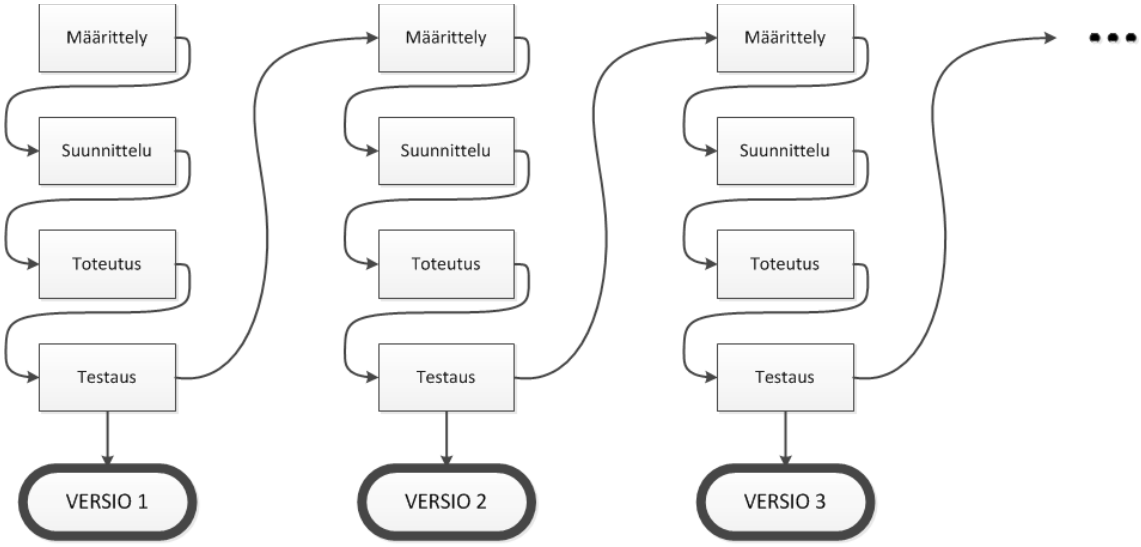
Kuva 7: Vesiputousmalli

3.6.3 Inkrementaaliset ja ketterät menetelmät

Inkrementaalisessa ohjelmistokehityksessä lisätään toimivaan ohjelmistoon ominaisuuksia ja toimintoja luoden aina uusia toimivia versioita. Ketterät menetelmät (Agile methods) perustuvat yleensä inkrementaaliseen kehitykseen ja ne korostavat mahdollisimman suoraa viestintää ja nopeaa reagointia muutostarpeisiin. Oleellista on todellinen avoin vuorovaikutus kehittäjien, muiden projektiin osallistuvien tahojen ja asiakkaan välillä. Menetelmät kannustavat ohjelmistokehittäjille annettaviin vapauksiin, mikä kannustaa itsenäiseen toimintaan ja lisää kehittäjän vastuuta toimivasta ohjelmistosta. Ketterät menetelmät korostavatkin usein kehittäjien osaamisen merkitystä kehitystyölle. Nykyisin tunnetuimpia menetelmiä ovat Scrum, XP-menetelmä (Extreme programming) ja Kanban. Myös useimmat ketterät menetelmät perustuvat kehityksen jakamiseen iteraatioihin, jotka yleensä ovat pituudeltaan viikosta kuukauteen. Ketterissä menetelmissä korostuu toimivan ohjelmiston merkitys, ja kattavan dokumentaation merkitys on vähäisempi kuin perinteisissä ohjelmistokehitysmalleissa.

Evo-malli (Evolution Delivery) on hyvä esimerkki inkrementaalisesta kehitysmallista, jossa ohjelmiston kehitys tapahtuu toistamalla vesiputousmallin mukaisia vaiheita peräkkäisinä ketjuina. Ideana on, että jokaisen kehityssyklin eli iteraation jälkeen tuloksena on toimiva järjestelmä, jota laajennetaan jälleen seuraavassa vesiputousmallin mukaisessa syklissä kuvan 8 mukaisesti. Syklien täytyy olla riittävän lyhyitä, tyypillisesti muutamasta päivästä pariin kuukauteen, jotta edellisen vaiheen tuloksia pystytään hyödyntämään nopeasti. Koska tällä mallilla toimien saadaan palautetta tehtyjen ratkaisujen toimivuudesta kehityksen aikana joka syklillä, voidaan nämä tarpeet huomioida seuraavassa syklissä. Koska ratkaisevien suunnittelulinjauksien

toimivuutta päästään testaamaan jo projektin alkupuolella, voidaan huonoja ratkaisuja korjata mahdollisimman aikaisin.

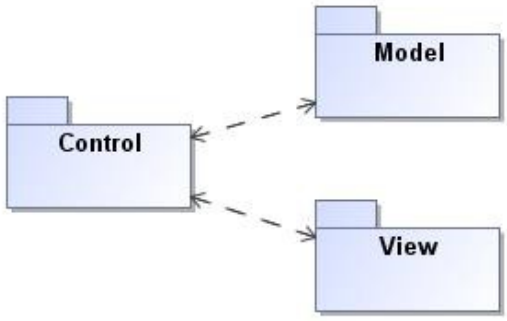


Kuva 8: EVO-malli

3.6.4 Suunnittelumallit

Suunnittelumallit ovat tärkeä työkalu nykyaikaisessa ohjelmistokehityksessä. Niiden avulla pyritään saavuttamaan ohjelmistolle selkeä rakenne ja näin helpottamaan ohjelmiston kehitystä ja ylläpitoa. Ne myös tarjoavat valmiita suunnitteluratkaisuja usein toistuviin ongelmiin. Ohjelmiston suunnittelu voidaan jakaa kolmeen osaan suunnnittelun laajuuden mukaan: arkkitehtuurisuunnitteluun, mekanistiseen suunnitteluun ja yksityiskohtaiseen suunnitteluun. (Kuikka 2007)

MVC-malli on yleisesti myös perinteisen ohjelmistotuotannon alalla käytetty suunnittelumalli. Sen periaatteena on erottaa varsinainen sovelluslogiikka mahdollisimman hyvin sitä ympäröivistä rajapinnoista, erityisesti käyttöliittymästä (Haikala & Märijärvi 2004). Mallin nimi tulee sanoista model, view ja controller. Model, eli malli tarkoittaa ohjelman sisältämää tietosisältöä, Inspectorin tapauksessa tietokantaan tallennettuja tietoja ja tietokannan rakennetta. View eli näkymä tarkoittaa käyttäjälle näkyvää osaa, käyttöliittymää. Controller eli toimintalogiikka ohjaa järjestelmän toimintaa kuvan 9 mukaisesti.



Kuva 9: MVC-malli

Model pitää sisällään kaikki esitettävät tiedot, tietokannat ja niiden muuttamiseen tarvittavien toimintojen rajapinnat. Inspectorin tapauksessa tietokanta tarjoaa mekanismit datan tallentamiseen tietokantaan ja tietojen lukemiseen käyttöliittymässä ja erilaisissa raporteissa esitettäväksi. Käyttöliittymä (View) on osakokonaisuus, joka näkyy käyttäjälle esimerkiksi selaimessa. Käyttöliittymä muodostuu dynaamisesti tilanteen mukaiseksi käyttäjän syötteiden ja tietokannan datan perusteella toimintalogiikan ohjaamana. Koska käyttöliittymä on erotettu sovelluslogiikasta, voidaan myöhemmin räätälöidä käyttöliittymiä erikseen esimerkiksi mobiililaitteille. Käyttöliittymä käyttää tietokantaosan tarjoamia palveluita toimintalogiikan välityksellä. Se esimerkiksi ohjaa käyttäjän antamat syötteet kontrolliosan käsiteltäväksi ja esittää tulosteet halutussa muodossa. Käyttöliittymä ei siis koskaan voi käsitellä suoraan tietokantaa, vaan tieto kulkee aina kontrollikerroksen kautta.

Toimintalogiikka (Controller) on nimensä mukaisesti ohjausosa, joka vastaa toimintojen suorituksesta ja järjestelmän toiminnasta. Inspectorissa toimintalogiikka myös huolehtii liitynnästä tietoa keräävään PLC:hen tarjoamalla PLC:lle rajapinnan, jonka kautta PLC lähettää datan tallennettavaksi tietokantaan. Kontrolliosa huolehtii esimerkiksi käyttöliittymältä tulevista syötteistä ja niiden pohjalta suoritettavien tietokantahakujen suorittamisesta. MVC-malli onkin todella yleinen graafisia käyttöliittymiä ja verkko-ohjelmointia sisältävässä ohjelmistokehityksessä (Ruohotie 2006).

3.6.5 Mallipohjainen kehitystyö

Mallipohjaisessa kehitystyössä pyritään hyödyntämään erilaisia ohjelmiston mallinnusmenetelmiä mahdollisimman tehokkaasti. Mallipohjaisessa kehitystyössä korostuu Douglasin mukaan alla luetellut hyviksi havaitut tekniikat.

- **Visuaalinen mallinnus:** Mallinnus mahdollistaa järjestelmän tehokkaan tarkistelun eri abstraktiotasoilla. Mallien avulla on mahdollista ymmärtää toimintoja ja mekanismeja hyvin karkealta järjestelmätasolta aina tarkkaan yksityiskohtatasoon saakka.

- **Suoritettavat mallit:** Suunniteltujen mallien testaaminen on työlästä ilman suoritettavia malleja tukevia työkaluja. Perinteisesti mallin pohjalta on jouduttu ohjelmoimaan kyseiset komponentit valmiiksi, jonka jälkeen niitä on vasta päästy testaamaan. Jos itse mallia voidaan jo testata mallin suunnitteluun käytetyllä työkalulla, säästetään merkittävästi aikaa ja työtä vikojen löytyessä jo ennen varsinaista ohjelmointia.
- **Mallin ja koodin liitännäisyys:** Merkittävä haaste ohjelmistosuunnittelussa käytettäessä visuaalisia malleja on, että malli ja toteutettu koodi vastaavat jatkuvasti toisiaan. Jos mallinnuksella ja koodin kirjoituksella ei ole mitään automaattista yhteyttä, muodostuu niiden sisältöihin helposti eroja kehityksen edetessä. Nykyään on saatavissa työkaluja, jotka integroivat mallinnustyökalut ja koodin dynaamisesti keskenään.
- **Automaattinen testaus vaatimusmäärittelyn pohjalta:** Kun ohjelmistoa kehitetään iteratiivisesti, on tuloksena prototyyppejä, jotka lisäävät toiminnallisuutta vähän kerrassaan aina edelliseen prototyyppiin. Kun prototyyppiin lisätyt uudet ominaisuudet testataan aina heti vaatimusten mukaan toimivaksi, on todennäköistä, että virheet korkealla abstraktiotasolla, esimerkiksi arkkitehtuurissa paljastuvat aikaisin.
- **Sovelluskehyksien käyttäminen:** Sovelluskehys (engl. Framework) tarjoaa alustan, jonka päälle voidaan kehittää sovelluskohtaiset ominaisuudet. Sevelluskehys pyrkii tarjoamaan valmiina ohjelmistojen yleiskäyttöiset komponentit, joita kannattaa hyödyntää aina kun mahdollista.
- **Iteratiivinen kehitys:** Merkittävä hyöty iteratiivisessa kehityksessä on mahdollisuus aikaisen vaiheen testaukseen. Iteratiivisella kehityksellä voidaan koostaa lopullinen sovellus laadukkaista ja toimiviksi todetuista ohjelmistokomponenteista.

(Douglass 2009)

Esimerkki mallipohjaista kehitystä toteuttavasta menetelmästä on ROPES-prosessi (Rapid Object-Oriented Process for Embedded Systems). Se on kehitetty laitteistonläheisen ohjelmoinnin tarpeisiin. ROPES soveltuu hyvin myös reaaliaikavaatimuksia sisältävien järjestelmien kehittämiseen ja menetelmä skaalautuu hyvin pienistä yhden hengen projekteista todella suuriin kehityshankkeisiin asti. Normaalien ohjelmistokehityksen tarpeiden lisäksi se pyrkii huomioimaan myös laitteiden suunnittelun vaatimukset, ohjelmiston niihin integroimisen ja kokonaisuuden testaamisen.

Olennaista ROPES-mallissa esiteltyjen tekniikoiden lisäksi ovat kehityssyklit ja niiden luokittelu ajan mukaan. Makrosykli kestää kuukausia tai vuosia ja se kattaa koko järjestelmän kehityksen valmiiksi asti. Mikrosyklit sen sijaan kestävät muutamia viikkoja ja pitävät sisällään yhden uuden prototyypin. Lyhyimpiä syklejä ovat

nanosyklit, jotka kestävät vain joitakin minuutteja. Nanosykleihin liittyy tuotoksien jatkuva testaaminen, jolloin nanosyklin aikana toteutetaan aina pieni muutos, joka testataan välittömästi. (Douglass 2009)

3.7 Tietoturva

Vaikka julkisessa keskustelussa tietoturvan merkitys tunnistetaan, on käytännön toteutuksissa huomattavan usein parantamisen varaa. Valitettavan usein kaikki tieto siirretään täysin salaamattomana, mikä altistaa järjestelmät tietojen kalastelulle tai vääristämiselle. Inspectoria kehitettäessä on testauksen ja kehitystyön helpottamiseksi käytetty salaamatonta tiedonsiirtoa, mutta jatkossa myös tietoturvaan ollaan kiinnittämässä huomiota. Tietoturvalle asettaa tiettyjä haasteita ohjelmoitavien logiikoiden laaja kirjo, välillä ohjauksen toimiessa täysiverisessä teollisuus-PC:ssa ja välillä laskentateholtaan ja muistiltaan hyvin rajoittuneessa pienessä logiikkaohjaimessa. Tässä kappaleessa kartoitetaan mahdollisia ratkaisuja tietojen salaamiseksi ja tietoturvan parantamiseksi.

3.7.1 Tietoturvan yleiset periaatteet

Tietoturvallisuus määritellään usein jakamalla se tekijöihin luottamuksellisuus, eheys ja käytettävyys. Luottamuksellisuus tarkoittaa sitä, että tietoa voivat käsitellä vain sellaiset tahot, joille on myönnetty siihen oikeus. Eheydellä tarkoitetaan tiedon muuttumattomuutta. Tieto ei siis saa muuttua vikojen, hyökkäyksien tai muiden tapahtumien seurauksena, ja mahdollisesti tapahtuneet muutokset täytyy pystyä luotettavasti havaitsemaan. Käytettävyys tarkoittaa tietoturvaa käsiteltäessä sitä, että tieto on siihen oikeutettujen saatavilla luotettavasti ja oikea-aikaisesti.

Tietoturvahyökkäykset voidaan jakaa kahteen luokkaan, aktiivisiin ja passiivisiin. Passiivisella hyökkäyksellä tarkoitetaan salakuuntelua ja tiedon päätymistä muiden kuin haluttujen tahojen tietoon. Aktiivinen hyökkäys tarkoittaa alkuperäisen tiedon muuntelua tai väärän tiedon luomista. Perushyökkäystyyppejä ovat keskeytys, sieppaaminen, muuntaminen ja väärentäminen. (Kerttula 1999)

3.7.2 Automaation tietoturva

Aikaisemmin automaatiojärjestelmät ovat olleet yleensä eristettyjä ulkoisista verkoista, minkä takia tietoturvaratkaisuihin ei ole tarvinnut kiinnittää suurta huomiota. Nykyään automaatiojärjestelmät ovat yhteydessä ulkoisiin tietoverkkoihin, joten myös tietoturva on noussut entistä tärkeämpään asemaan. Automaatioverkot käyttävät nykyään usein samoja tekniikoita ja osittain samoja standardeja kuin muutkin tietoverkot. Tästä seuraa se, että verkkoihin liittyvät vaarat uhkaavat myös automaatiojärjestelmiä, mutta toisaalta myös mahdollistavat vakiintuneiden ja toimiviksi todennettujen tietoturvaratkaisujen käyttämisen. Automaatiojärjestelmien verkkoihin kohdistuu myös erikoisvaatimuksia

järjestelmien luonteesta johtuen, kuten esimerkiksi yhteys muihin turvallisuuden kannalta kriittisiin järjestelmiin, mahdolliset tiukat reaaliaikavaatimukset ja komponenttien suorituskyvyn kriittisyys.

Automaatiojärjestelmien tietoturvaan tuo lisähaastetta myös järjestelmien pitkä elinikä. Kun normaalien IT-komponenttien elinkaari on yleensä muutamia vuosia, voi automaatiojärjestelmä olla käytössä useita vuosikymmeniä. Näin ollen käytössä olevien komponenttien tuki, ylläpito ja korvattavuus voivat olla puutteellista. Järjestelmän ollessa vanha, mutta todella kompleksinen, ei tietoturvaominaisuuksien vaatimuksia luultavasti ole järjestelmää suunniteltaessa otettu huomioon. Näistä syystä järjestelmien päivittäminen uusilla komponenteilla tai tietoturvaominaisuuksilla voi olla todella haasteellista tai jopa mahdotonta.

3.7.3 Tiedon suojausmenetelmät

Tieto voidaan suojata salaamalla eli kryptaamalla se, eli muuttamalla sen muotoa siten, että tieto ei ole luettavissa purkamatta salausta. Tiedon kryptaus voidaan toteuttaa symmetrisellä tai asymmetrisellä algoritmilla.

Symmetrinen salausalgoritmi käyttää viestin salaamiseen ja purkamiseen samaa avainta. Käytettäessä symmetristä algoritmia viestin salaaminen ja purkaminen ovat kohtalaisen yksinkertaisia operaatioita, millä saavutetaan suuri suorituskyky pienemmällä laskentateholla ja muistinkulutuksella. Käytännössä algoritmi sekoittaa viestiä bittitasolla algoritmin ja sovitun avaimen mukaan. Suurimpana ongelmana symmetrisillä menetelmillä on avaimen lähettäminen toiselle osapuolelle ennen salatun yhteyden muodostumista.

Asymmetrisessä eli epäsymmetrisessä salauksessa salaukseen ja viestin purkamiseen käytetään eri avaimia. Menetelmässä käytetään kahta eri avainta, joista toinen on julkinen ja toinen on salainen. Viesti salataan vastaanottajan tarjoamalla julkisella avaimella, joka on nimensä mukaisesti muiden nähtävillä. Kun viesti on salattu, sen avaamiseen tarvitaan viestin vastaanottajan hallussa oleva salainen avain. Näin ennestään tuntemattomat osapuolet voivat luoda välilleen suojatun yhteyden vaihtamalla ensin julkisia avaimia avoimen verkon yli, minkä jälkeen kumpaankin suuntaan lähetettävät viestit pystytään salaamaan. Julkisen avaimen menetelmällä tehtävä salaus vaatii osapuolilta runsaasti laskentatehoa ja aiheuttaa muistinkulutusta, joka saattaa olla ongelma varsinkin laitteistonläheisessä ohjelmoinnissa. Useat nykyaikaiset salausmenetelmät ovatkin yhdistelmiä symmetrisestä ja epäsymmetrisestä menetelmästä, jolloin saavutetaan epäsymmetrisen menetelmän joustavuus menettämättä symmetrisen menetelmän suorituskykyä.

3.7.4 IPSec

IPSec (IP Security Architecture) on protokollaperhe, jonka avulla pystytään toteuttamaan tarvittava tietoturva IP-protokollaa käytettäessä. IPSec:n määrittelemät protokollat voidaan jakaa kahteen luokkaan, pakettivirtojen turvaamiseen ja avaintenvaihdolla turvattujen yhteyksien muodostamiseen tarkoitettuihin protokolliin. Pakettivirtojen suojaamiseen on IPSec:ssa kaksi eri vaihtoehtoa. EPS (Encapsulating Security Payload) mahdollistaa pakettivirran salaamisen ja harvemmin käytetty AH (Authenticating Headers) mahdollistaa todennuksen ja takaa viestin eheyden, mutta ei takaa luottamuksellisuutta.

AH-protokolla käyttää tiedon eheyden varmistamiseen niin sanottua HMAC-koodia. Lisäksi jokaisella yhteydellä on yksilöllinen tunnistenumero (SPI, Security Parameter Index), jolla kyseiseen yhteyteen kohdistuvat paketit voidaan tunnistaa. AH-protokolla toteuttaa myös sekvenssinumeron, jonka arvo kasvaa joka viestissä, ja jonka avulla voidaan varautua kaapattujen pakettien uudelleenlähettämiseen. Nämä tiedot lisätään AH-otsakkeeseen, joka lisätään siirrettävään pakettiin. Myös EPS varmistaa tiedon eheyden HMAC:lla. EPS-otsakkeesta löytyvät myös SPI-numero sekä sekvenssinumero. Lisäksi tieto salataan valitulla salausalgoritmilla, jolloin varmistutaan myös tiedon luottamuksellisuudesta.

IPSec:a voidaan käyttää kuljetus- tai tunnelointimuodoissa. Kuljetusmuodolla tarkoitetaan kahden laitteen välisen tiedonsiirron suojaamista yhteyden päästä päähän. Tunneloinnilla taas tarkoitetaan sitä, että alkuperäinen IP-paketti kuljetetaan tuntemattoman verkon osan läpi turvatussa tunnelissa, jota varten siirrettävä IP-paketti kääritään uuden paketin sisälle ja annetaan sille uusi IP-otsake.

Avainten hallinta voidaan toteuttaa manuaalisesti tai se voidaan automatisoida. Manuaalinen tapa on työläs ja altis virheille, joten on syytä suosia automatisoitua avainten hallintaa, jos vain mahdollista. Automatisoidussa avaintenvaihdossa käytetään avaintenvaihtoprotokollana useimmiten IKE-protokollaa (Internet Key Exchange).

IPSec on hyvin yleisesti käytetty menetelmä. Useimmiten IPSec:a käytetään muodostettaessa VPN (Virtual Private Network) yhteyksiä, jolloin kaksi eri tietoverkkoa yhdistetään toisiinsa IPSec tunnelin avulla. IPSec on toteutettu myös joillekin sulautetuille järjestelmille, joten sitä on mahdollista soveltaa myös laitteistonläheisessä ohjelmoinnissa.

3.7.5 TLS

TLS-salausprotokolla on TCP-yhteyden salaamiseen käytetty standardi, josta käytettiin aikaisemmin nimeä SSL (Secure Sockets Layer). Protokolla perustuu epäsymmetrisellä menetelmällä tehtävään kättelyyn. Kättelyn aikana osapuolet autentikoidaan käyttämällä sertifikaatteja ja lisäksi neuvotellaan myöhemmin käytettävä symmetrinen

salausmenetelmä. Kättelyn jälkeen siirrettävä tieto salataan sovitulla symmetrisellä algoritmilla ja avaimella. (RFC5246 2015)

TLS-protokollaa voidaan käyttää yhdessä HTTP-protokollan kanssa, jolloin tiedot salataan TLS-menetelmällä ennen lähettämistä. Tästä käytetään nimitystä HTTPS (Hypertext Transfer Protocol Secure), ja sen käyttö on nykyään varsin yleistä käytettäessä esimerkiksi salaamista tarvitsevia verkkosivuja.

TLS käyttää TCP-protokollaa, joten laitteistonläheisen ohjelmoinnin vaatimuksena on kyseisen protokollapinon saatavuus käytettävälle laitteelle. Automaatiojärjestelmän tietoturvaa toteutettaessa on myös hyvä ottaa huomioon, että TLS vaatii paljon laskentatehoa ja kuormittaa suoritinta yhteyttä muodostettaessa. Yhteyden muodostamisen jälkeen kuormitus kuitenkin pienenee huomattavasti, joten yhteyden muodostuksen aiheuttama kuormitushuippu on hyvä huomioida tietoturvaa suunniteltaessa sulautetuille järjestelmille.

3.7.6 AES

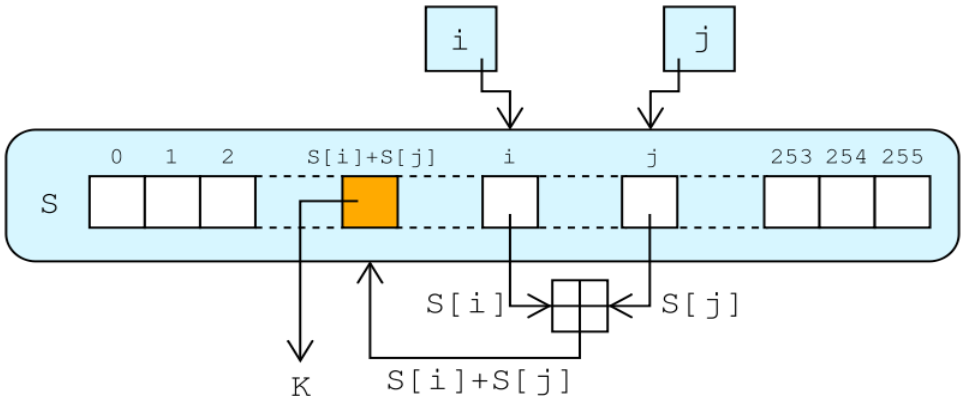
AES (Advanced Encryption Standard), joka on alkuperäiseltä nimeltään Rijndael, on symmetriseen algoritmiin perustuva salausmenetelmä. AES-salausalgoritmi käyttää 128, 192 tai 256 bitin pituista salausavainta. AES on varsin yleisesti käytetty salausalgoritmi, ja sillä saavutettavan vahvan salauksen vuoksi se on valittu esimerkiksi USA:n viranomaisten käyttöön viralliseksi salausmenetelmäksi. (Järvinen 2003)

AES ei vaadi suuria resursseja salausalgoritmin ollessa symmetrinen, joten se soveltuu hyvin myös sulautetuissa järjestelmissä käytettäväksi. Menetelmällä saavutettu salaus on kuitenkin todella tehokas, joten AES:lla tehty salauksen taso riittää varmasti myös automaatiojärjestelmien salaukseksi. AES standardin mukaisia toteutuksia löytyykin jonkin verran sekä kaupallisina että ilmaisina myös joillekin sulautetuille järjestelmille. Valitettavasti AES-toteutusta ei löytynyt kuitenkaan IEC-61331-3 standardia noudattavalla PLC:lla sovellettavaksi, joten AES-standardia käytettäessä PLC:lla täytyisi se toteuttaa itse, mikä vaatisi paljon ohjelmointia ja testausta.

3.7.7 RC4

RC4-algoritmi oli aikanaan kaupallisia tarkoituksia varten suojattu algoritmi, mutta sen vuodettua julkisuuten se yleistyi myös ARC4 nimellä tunnettuna. RC4 on yksinkertainen symmetrinen salausmenetelmä. RC4:lle on ominaista sen todella helppo toteutettavuus koodissa. Yksinkertaisen toteutuksen haittana on muutamat havaitut puutteet salausalgoritmissa. Näiden puutteiden korjaamiseksi on kehitetty paranneltuja versioita algoritmista, mutta täysin aukoton niidenkään tarjoama tietoturva ei silti ole. RC4 on ollut aikaisemmin käytettävissä myös TLS-salausprotokollassa, mutta löydettyjen puutteiden vuoksi se ei enää ole käytettävissä TLS:n kanssa.

Algoritmin perustana on tuottaa suuri määrä satunnaislukuja sovitun salausavaimen perusteella. Satunnaislukujen ja salattavan tekstin kanssa suoritetaan XOR-operaatioita, joista muodostuu salattu teksti. Samalla yksinkertaisella algoritmilla onnistuu myös viestin purkaminen, koska seuraava XOR-operaatio samoja tietoalkioita käyttäen kumoaa ensimmäisen. Helppo tapa vaikeuttaa RC4-salauksen murtamista on jättää salausavaimella generoidun avainvirran ensimmäisistä alkioista määrätty osa käyttämättä. Salatun tuloksen muodostusta havainnollistaa kuva 10. RC4 algoritmin toteutus on myös tämän työn liitteenä 1 C#-kielellä toteutettuna.



Kuva 10: RC4-algoritmin toimintaperiaate

3.7.8 OPC UA:n tietoturva

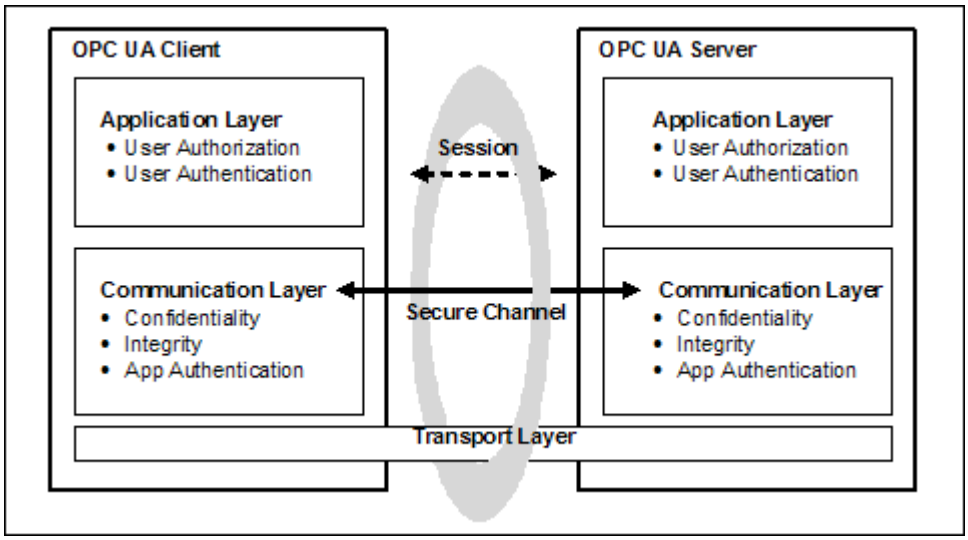
OPC UA:ssa on tietoturvan merkitys huomioitu protokollaa suunniteltaessa. Standardi pitääkin sisällään vahvat tietoturvaominaisuudet. OPC UA määrittelee kaksi tietoturvaprotokollaa sovelluksien käytettäväksi, WS-SecureConversation ja UA-SecureConversation. WS-SecureConversation on tarkoitettu xml-muotoisen tiedon suojaamiseen käytettäessä sovelluspalveluita. WS-SecurityConversationia, joka koostuu useista eri menetelmistä ja teknologioista, ylläpitää OASIS (Organization for the Advancement of Structured Information Standards). Se pitää sisällään esimerkiksi standardit XML-muotoisen datan salaamiseksi ja allekirjoittamiseksi. UA-SecureConversation on kehitetty erityisesti OPC UA:ta varten. Se yhdistää hyviksi havaittuja menetelmiä WS-SecureConversationista ja TLS-protokollasta, jotka on muokattu vastaamaan OPC UA:n tarpeisiin. (Mahnke et al. 20009)

OPC UA salaa siirrettävät tiedot käyttäen joko 128- tai 256-bittistä salausta, jolla taataan tiedon luottamuksellisuus. Protokolla varmistaa myös tiedon eheyden ja muuttumattomuuden viestien allekirjoitusmekanismillaan ja viestien sekvenssinumeroilla pystytään välttämään viestien uudelleenlähetykseen perustuvat hyökkäykset. OPC UA autentikoi tietoliikenteen molemmat osapuolet käyttäen

OpenSSL sertifikaatteja. Autentikoinnin perusteella käyttäjäoikeuksia voidaan hallita ja eri osapuolien toimia voidaan sallia tai evätä halutulla tavalla. (OPC Foundation 2015)

Yhteyden muodostukseen kuuluvat kättelyt kuluttavat resursseja, joten uutta yhteyttä ei ole järkevää luoda jokaiselle tiedonsiirtotapahtumalle erikseen. Sen sijaan OPC UA sallii tietoturvallisen istunnon avoinna pitämisen vaikka ikuisesti, joten resursseja kuluttava yhteyden muodostus voidaan tehdä esimerkiksi vain laitteen käynnistyksen yhteydessä.

OPC UA:n määrittelemä tietoturvamalli jakaa toteutettavat tietoturvaominaisuudet eri kerroksiin kuvan 11 mukaisesti. Mallissa esitetty kommunikointikerros sisältää mekanismit, joilla varmistetaan tiedon salaamisella saavutettava luottamuksellisuus, viestien allekirjoituksilla varmistettava tiedon eheys ja sertifikaateilla todettu sovellusten autentikointi. Tällä kerroksella OPC UA palvelin ja asiakas neuvottelevat käytettävät tietoturvaprotokollat ja näin muodostavat suojatun yhteyden. Sovelluksen käsittelemän tiedon käsittely ja lähetys suoritetaan sovelluskerroksella. Tämän sovelluslogiikan lisäksi kyseinen kerros vastaa myös käyttäjän tunnistuksesta ja käyttäjäoikeuksien hallinnasta. Sovellus siis avaa istunnon käyttämällä kommunikointikerroksen tarjoamaa suojattua yhteyttä ja muodostaa siirrettävän tietosisällön. Kommunikointikerros salakirjoittaa ja paketoi tiedon ja välittää sen siirtokerrokselle, joka huolehtii datapaketin siirtämisestä verkon yli. (Renjie et al. 2015)



Kuva 11: OPC UA Tietoturvamalli

4 TIEDONKERUUJÄRJESTELMÄLLE ASETETUT VAATIMUKSET

Järjestelmän tehtävänä on toteuttaa sivulla 4 esitetyn kuvan 2 mukainen toiminnallisuus. Koska ohjelman toteutus oli jo pitkällä ja useita toimivia komponentteja oli jo käytettävissä, eivät kehitysympäristöjen ja laitteiden radikaalit muutokset olleet enää tätä työtä tehdessä toivottuja.

Inspectoria käytetään toistaiseksi vain tiedonkeruuseen, eikä sillä ole näin ollen ohjausteknisiä reaaliaikavaatimuksia. Mikään ei kuitenkaan estä yhdistämästä myös laiteohjauksia suoritettavaksi samalla kontrollerilla osana Inspectoria, joten Inspectoria suunniteltaessa ja toteutettaessa on pyritty ottamaan automaation ohjelmistojen erityispiirteet mahdollisimman hyvin huomioon.

4.1 Kehitysympäristö ja käytettävät laitteistot

PLC:n ja pilvipalvelun tiedonsiirron rajapinnan määrittely mahdollistaa myöhemmin toisen osapuolen toteutusyksityiskohtien ja laitteiston muuttamisen rajapinnan pysyessä entisellään. Käytettävät ympäristöt on kuitenkin pyritty valitsemaan siten, että tarvetta muutokselle ei lähitulevaisuudessa tulisi.

4.1.1 Logiikkaohjelmointiympäristö

PLC:n ohjelmointiin käytetään Beckhoff-logiikkavalmistajan kehitysympäristöä Twincat. PLC kehitysympäristö koostuu kahdesta oleellisesta osiosta, *System Managaer* ja *PLC Control*. System Manageria käytetään kontrollerin konfiguroimiseen. Sen avulla määritellään linkit ohjelmamuuttujien ja fyysisten laitteiden välillä. Lisäksi sitä käyttämällä voidaan määritellä sekä muokata kontrollerin perusasetuksia, kuten verkko-osoitteita ja reaaliaikaominaisuuksia.

Toinen oleellinen osa Twincat kehitysympäristöä on PLC Control ohjelmisto. Se sisältää työkalut ohjelmien kirjoittamiseen. Ohjelmointia voi suorittaa kaikilla IEC-61131-3 standardin määrittelemillä kielillä, joista Inspectorin ohjelmointi on tehty lähes kokonaan ST-kielellä. PLC Control ohjelmaa käytetään myös ohjelman siirtämiseen logiikalle ja sillä voi seurata ohjelman reaaliaikaista suoritusta tai suorittaa ohjelmaa rivi kerrallaan.

InSolutionilla on ollut pitkään projekteissaan käytössä Twincatin versio 2, jolla myös Inspectoria tällä hetkellä kehitetään. Twincat kehitysympäristöstä on nykyään saatavilla myös uudempi versio 3. Sitä ei kuitenkaan vielä ole otettu Inspectorin kehityksessä käyttöön aiemmin kehitetyn koodin siirrettävyysongelmien vuoksi ja vuosien saatossa kertyneen vahvan Twincat 2 kokemuksen vuoksi. Vaikka Twincat 2 onkin edelleen

täysin toimiva kokonaisuus alkaa se olla selkeästi elinkaarensa loppupuolella, joten siirtyminen Twincat 3:een kannattaisikin tehdä mahdollisimman pian.

4.1.2 Käytetty logiikkakontrolleri CX8090

Logiikkakontrolleriksi on valittu Beckhoffin CX8090. Kyseinen malli on edullinen, mutta pitää sisällään kaikki tarvittavat ominaisuudet. CX-tuotesarja käyttää Windows CE käyttöjärjestelmää ja Twincatin ajo suoritetaan niin sanottuna soft-PLC:na. Inspectorin PLC:lta ei oletettavasti tarvita suurta suoritustehoa, vaikka datamäärät saattavatkin kasvaa suuriksi, koska logiikalla ei ole tarkoitus tehdä vaativaa laskentaa. Vaikka tuotettu koodi itsessään on täysin yhteensopivaa eri kontrollerien välillä, vaativat eri laitteistot kuitenkin testaamista. Käytännössä esimerkiksi TCP/IP-protokollan toiminnassa on aikaisemmin huomattu eroja halvimpien mallien toteuttaessa protokollan natiivisti laitetasolla, kun taas tehokkaammat mallit käyttävät Windows CE käyttöjärjestelmän erillistä TCP/IP kirjastoa.

Kuva 12: Logiikkakontrolleri

CX8090:ssa on sisäistä keskusmuistia 64 MB ja suorittimena siinä käytetään 32 bittistä ja 400 Mhz kellotaajudella toimivaa suoritinta. Kontrolleria voidaan ohjelmoida ja käyttää ethernetliitynnän avulla. Kontrollerista löytyy myös sisäinen paristolla varmistettu kello. Maksimi tehonkulutus on 3W ja laite on passiivisesti jäähdytetty, eli se ei sisällä tuulettimia tai muita mekaanisia helposti teollisuusympäristössä vikaantuvia komponentteja. Massamuistina kontrolleriin kuuluu microSD muistikortti, jonka koko on vakiokokoonpanossa 512 MB, mutta se on mahdollista vaihtaa suurempaan korttiin. Keskusmuistissa on ajon aikana tallessa niin suoritettava käyttöjärjestelmä, kuin PLC

ohjelma ja sen käyttämät tiedotkin. Koska keskusmuisti tyhjennetään aina virran katkaisun yhteydessä, ovat tiedot tallessa massamuistissa. PLC-ohjelman pysyväismuistiksi määritellyt muuttujat tallennetaan muistikortille virran katkaisun yhteydessä, mutta muistikortin koko ei ole ainoa PLC-ohjelman puskurointikykyä rajoittava tekijä. PLC:ssa on sisäinen UPS, joka vastaa virransyötöstä kontrollerille vielä noin sekunnin ulkoisen käyttöjännitteen lakattua, jolloin tiedot kirjoitetaan muistikortille. Massamuistille kirjoittaminen on kuitenkin hidasta, joten maksimimäärä pysyville muuttujille on 1 MB.

4.1.3 Palvelinympäristö

Palvelimen ohjelmisto on kehitetty Microsoftin .NET ympäristöä käyttäen. Tämä mahdollistaa palvelimen ja käyttöliittymien ylläpidon ja laajennettavuuden pitkälle tulevaisuuteen. Tällä hetkellä käyttöliittymä on rakennettu pääasiassa verkkoselaimessa toimivaksi, mutta ohjelmisto ollaan toteuttamassa myös ainakin Android mobiilialustalle. Tietokantana käytetään SQL tietokantaa, jolle löytyy runsaasti ilmaisia työkaluja ja joka on todella monipuolinen ja yleisesti käytetty relaatiotietokantajärjestelmä.

4.2 Käyttäjät

Käyttäjien tunnistus on toteutettu salasanalla. Toistaiseksi käyttäjätasoja on vain yksi, mutta tulevaisuudessa tullaan toteuttamaan eri käyttäjätasot, jolloin esimerkiksi mittauksen parametreja pääsevät muuttamaan vain pääkäyttäjät, mutta muut pystyvät silti seuraamaan mittaustuloksia. Käyttäjätasojen toteutuksesta vastaavat käyttöliittymä ja palvelin, joten PLC-muutoksia ei käyttäjätilien hallinnan kehityksestä johtuen ole odotettavissa.

4.3 Toiminnot

PLC-ohjelmalla on selkeästi kolme eri toimintoa. Käynnistyessään ohjelman on suoritettava tarpeelliset alustukset ja valmistauduttava jatkuvan mittauksen suorittamiseen. Ohjelman täytyy myös pystyä muuttamaan määritettyjä parametreja palvelimelta saadun parametriviestin perusteella. Näiden lisäksi kolmas toiminto on jatkuva mittaus ja mittaustiedon lähetys, jota PLC suorittaa niin kauan kuin PLC on käynnissä.

4.3.1 Käynnistys

PLC-ohjelma käynnistyy automaattisesti kun kontrollerin käyttöjännite kytketään päälle. Ohjelman on pystyttävä omatoimisesti aloittamaan mittaus määritellyllä tavalla. Käynnistyessään ohjelma alustaa tarvittavat muuttujansa ja saattaa itsensä

käynnistyksen aikana toimintavalmiiksi. Lisäksi ohjelman on pystyttävä muodostamaan yhteys palvelimeen ja aloitettava mittaustietojen siirto palvelimelle.

4.3.2 Parametrien muutos

Käyttöliittymästä täytyy olla mahdollista muuttaa tiettyjä PLC:n parametreja, kuten mitattavalle signaalille suoritettavan prosessoinnin parametreja. Palvelimelta täytyy siis olla saatavilla parametriviesti, joka voidaan käsitellä PLC:lla ja jolla kyseiset parametrit voidaan muuttaa halutuiksi. Muutetut parametrit tallennetaan PLC:lle siten, että ne pysyvät tallessa myös sähkökatkojen yli.

4.3.3 Jatkuva mittaus

Tietoa kerätään ja lähetetään palvelimelle kolmessa eri muodossa. Nimitystä ADC-signaali (Automatic Data Collection) käytetään tässä sovelluksessa mittaus- ja tiedonsiirtotavasta, jossa lähetetään tiedot vain tilojen muutoksista. Hyvä esimerkki tällaisesta tiedosta on käyntitiedon seuranta, jossa PLC saa laitteelta binäärisen päällä/pois-tiedon. Palvelimelle lähetetään tällöin tieto vain tilan muutoksista sekä tiedot ohjelman käynnistyessä, jolloin datan siirron tarve vähenee. Koska PLC:n sammuminen tai yhteyden pitkäaikainen katkeaminen täytyy kuitenkin erottaa normaalista toiminnasta, jossa ei tapahdu tilamuutoksia, lähettää PLC palvelimelle määrätyin väliajoin niin sanottua heartbeat signaalia.

Toinen toiminto on nimetty CM-tiedoksi, jota voidaan käyttää esimerkiksi kunnonvalvontaan (Condition Monitoring). Mitattava tieto voi olla esimerkiksi lämpötila tai analoginen arvo tärinän voimakkuudesta. Raakadatan mittaus suoritetaan jokaisella PLC-syklillä, jonka pituus on tässä esimerkkisovelluksessa 50ms, mutta se on helposti muutettavissa sovelluskohtaisten vaatimusten mukaan. Raakadatalle suoritetaan jo mittausvaiheessa parametroitava alipäästösuodatus. Saatu mittaustieto lähetetään palvelimelle aina määritetyn ajan välein, esimerkiksi kerran kymmenessä sekunnissa. Lisäksi mitattavalle suureelle lasketaan normaalia mittausta pidemmässä käyttäjän määrittämässä syklissä minimi-, maksimi- ja keskiarvoa.

Kolmas siirrettävä tiedon laji on tapahtumat (engl. Event). Tapahtumat voivat olla esimerkiksi tieto virtakatkosta tai mitattavalle suureelle määritetyn maksimiarvon ylittyminen. Tällaisissa tilanteissa tieto tapahtumasta välitetään palvelimelle. Poikkeavat tapahtumat mitattavassa signaalissa vaikuttavat palvelimelle lähetettävän tapahtumatiedon lisäksi lähetettävään CM-dataan. Jos mittaustulos on käyttäjän määrittämän vaihteluvälin ulkopuolella, ohjelma lähettää palvelulle suuremmalla näytteistystaajuudella mittaukset tapahtumaa edeltävältä ja sitä seuraavalta ajalta. Puskuroitua raakadataa tulee siis säilyttää määritetty aika puskurissa mahdollisten poikkeustilanteiden varalta, ennen kuin ne voidaan ylikirjoittaa.

4.4 Ohjelmiston suoritusvaatimukset

Ohjelmistolle asetetut suoritusvaatimukset voidaan jakaa kahteen osaan, mittauksen suorittamiseen ja tuloksien tallentamiseen sekä tuloksien hallintaan ja käyttäjälle esittämiseen. Mittaustuloksien hallinta ja käyttäjälle esittäminen perustuvat yksinomaan tietokannan ja käyttöliittymän väliseen toimintaan, joten sen käsittely jätetään tämän työn ulkopuolelle. Mittauksen suorittamiselle ja tuloksien tallentamiselle sen sijaan on hyvä määrittää reunaehtoja. Normaalissa käytössä pienin tarvittava näytteistystaajuus on 1/s. Tämän mittaaminen ei tule olemaan ongelma, koska kokemuksesta PLC:n tiedetään sovelluksen laajuus huomioon ottaen pystyvän ilman optimointeja muutaman millisekuntin syklinaikaan. Sen sijaan tiedon väliaikaiseen tallentamiseen käytettävien rakenteiden muistinkulutus määrittää rajat puskuroitavalle datalle ja yhteyskatkojen maksimipituudelle.

4.4.1 Tiedon puskurointi

Sovelluksen toimiessa verkkoyhteyksien varassa on selvää, että välillä yhteydessä ilmenee ongelmia. Yhteysongelman ilmetessä on tärkeää, että PLC ei hukkaa dataa, ja toipuu yhteyskatkosta automaattisesti yhteyden alettua taas toimia. TCP-protokollan mukaisesti palvelin vahvistaa jokaisen datapaketin vastaanoton PLC:lle. Tällä varmistetaan datan perillemeno tai mahdollinen uudelleenlähetyksen tarve. Yksinkertaistettuna datan yhtenäisyyden periaatteena on monotoninen kirjoitus, joka tarkoittaa että lähetetty tietoalkio poistetaan ja seuraavat tietoalkiot lähetetään vasta, kun edellisen paketin lähetys on onnistunut (Tanenbaum & van Steen 2007). Puskurin maksimikokoa rajoittaa logiikkakontrollerin pysyvän muistin suurin sallittu koko. Suurinta sallittua yhteyskatkon pituutta määrittää puskurin koon lisäksi näytteistystaajuus ja mitattavien suureiden lukumäärä.

4.4.2 Datan tallennus häiriötilanteissa

Datan häviön tulisi olla mahdollisimman pieni myös sähkökatkossa ja PLC:n uudelleenkäynnistyksessä. Tästä syystä PLC:n tallentamat mittaustulokset, joita ei ole vielä siirretty onnistuneesti palvelimelle, tulee säilyttää pysyväismuistialueella. Pysyväismuistialueen tiedot pysyvät muuttumattomina sähkökatkon yli, jolloin yhteyden muodostusta ja tietojen lähetystä voidaan yrittää taas heti PLC:n käynnistyttyä, jolloin menetetään mittaustiedot ainoastaan virtakatkon ajalta. PLC-ohjelman tulee siis pystyä käyttöjännitteen palautumisen jälkeen automaattisesti muodostamaan yhteys, jatkamaan mittaamista ja siirtämään ennen ja jälkeen sähkökatkon tallennetut mittaustulokset palvelimelle.

5 OHJELMISTON TOTEUTUS

Tätä työtä tehdessä oli osa sekä palvelinohjelmistoa, että myös PLC-ohjelmaa jo melko pitkälle toteutettu. Näin ollen tämän työn puitteissa merkittäviä arkkitehtuurimuutoksia ei enää ollut tehtävissä, vaan painopiste oli ohjelmiston jatkokehittämisessä vaadittujen ominaisuuksien osalta.

5.1 Toiminnot

PLC-ohjelman suorittamat toiminnot on kuvattu kappaleessa 4.3. Nämä toiminnot eivät vaadi käyttäjältä erillisiä toimenpiteitä, pelkkä PLC:n käyttöjännitteen ja verkkoyhteyden kytkeminen riittävät.

5.1.1 Automaattinen alustus

Osa PLC:n muistista voidaan määritellä luonteeltaan pysyväismuistiksi. Tälle muistialueelle tallennetaan kaikki mittaamiseen, yhteyden muodostukseen ja ohjelman toimintaan muutoin liittyvät parametrit. Käytännössä CX-mallinen logiikkakontrolleri tallentaa pysyviksi määriteltyjen muuttujien arvot tiedostoon aina kun PLC sammutetaan. Käynnistyessään PLC jälleen lukee muuttujien arvot. Myös puskurirakenne, johon mittaustiedot tallennetaan mittauksen jälkeen odottamaan lähetystä, on määritelty pysyväismuistin alueelle. Näin ollen myöskään jo mitattua, mutta toistaiseksi palvelimelle siirtämätöntä tietoa ei pääse virtakatkon sattuessa katoamaan. Vastaavasta PLC:lle tallennetusta tiedostosta kontrolleri lukee käynnistyessään myös suorittamansa ohjelman, joka käynnisty automaattisesti.

5.1.2 Parametrien muutos

Parametrien muuttaminen on toteutettu siten, että PLC kysyy mahdollisia parametrimuutoksia palvelimelta säännöllisin väliajoin määritetyllä JSON viestillä. Kyseinen toteutustapa on tiedonsiirron tehokkuuden kannalta huono, mutta kyseisellä arkkitehtuurilla parhaaksi ratkaisuksi tässä vaiheessa nähty. Koska PLC huolehtii aina yhteyden luomisesta tiedonsiirtoa varten, ei palvelimen ole mahdollista lähettää parametrimuutoksia logiikalle ilman PLC:lta tulevaa pyyntöä. Pyynnön tullessa palvelin lähettää muuttuneet parametrit PLC:lle vastausviestissä.

5.1.3 Jatkuvat toiminnot

CM-signaali tallennetaan rengaspuskuriin, josta näytteet voidaan lähettää palvelimelle normaalia suuremmalla näytteistystaajuudella poikkeavan tapahtuman sattuessa. Näiden mahdollisten tietopurskeiden lisäksi mitattavasta suureesta lasketaan ja lähetetään keskiarvo sekä minimi- ja maksimiarvot parametroiduin väliajoin. Lähetystä varten on

oma tietorakenne, jonne lähetettävät tiedot puskuroidaan. Kaikki kyseiseen CM-signaaliin liittyvät tiedot tallennetaan samaan lähetyspuskuriin ja jokaiselle mitattavalle suureelle on oma puskurinsa. Myös jokaiselle seurattavalle ADC-tiedolle on oma puskurinsa, jonne siis tallennetaan kaikki tilamuutokset. Lisäksi oma puskuri on myös tapahtumille, jonka kautta lähetetään tapahtumatiedot ja parametrien pyynnöt palvelimelle. Lähetyspuskurit ovat tyypiltään taulukoita ja taulukon jokainen data-alkio pitää sisällään tarvittavat tiedot kuten aikaleiman ja mittauksen arvon.

Jokaiselle signaalityypille on oma kommunikointia hallinnoiva toimilohko. Kommunikointilohko tyhjentää lähetyspuskuria aina, kun siellä on lähetystä odottavia tietoalkioita. Kommunikointia varten lähetyspuskurin alkioista muodostetaan JSON-tietoalkioita, jotka kääritään http-kehykseen ja lähetetään palvelimelle. Kun siirto on onnistunut, poistetaan kyseiset tiedot lähetyspuskurista.

Kontrollerin ominaisuuksista johtuen pysyväismuistiin tallennettavien tietojen maksimikoko on 1 MB. Järjestelmän käyttämä pysyväismuisti koostuu lähes kokonaan mittaustiedoista, muun pysyvästi talletettavan tiedon ollessa suuruusluokaltaan muutamia satoja tavuja. Yksittäisen mittaustuloksen kuluttama muistin määrä on 20 tavua, joten kaikkiaan puskureihin mahtuvien mittausten määrä on noin 50000 tieto alkiota. Tältä pohjalta voidaan helposti laskea pisin mahdollinen yhteyskatko ilman tietohäviöitä, kun huomioidaan projektikohtaiset parametrit, kuten näytteistystaajuus ja mitattavien signaalien lukumäärä CM-signaaleille. ADC-tieto on muistinkulutukseltaan huomattavasti tehokkaampaa, koska siihen tallennetaan vain tilan muutokset. Mitattavan järjestelmän ominaisuudet vaikuttavat kuitenkin paljon ADC-mittauksen tuloksina oleviin datamääriin, joten niiden arvioiminen on hankalaa.

5.1.4 Tietoturvatoteutus

Tätä työtä tehdessä on muodostunut vahva näkemys siitä, että tiedonsiirrossa kannattaa tulevaisuudessa siirtyä käyttämään OPC UA standardia, jota ei ollut kuitenkaan mahdollista implementoida tämän työn puitteissa. OPC UA tulee myöhemmin tuomaan standardin mukaisen vahvan tietoturvan sisäänrakennettuna. Lisäksi PLC:lle toteutettuja käyttökelpoisia avoimia tietoturvatoteutuksia ei löytynyt. Tästä syystä päätettiin toistaiseksi toteuttaa tietoturva helposti toteutettavissa olevalla RC4-menetelmällä. RC4 ei tarjoa parasta mahdollista tietoturvaa, mutta sen tarjoama tietoturvataso riittää toistaiseksi kyseisen sovelluksen tarpeisiin. Kuitenkin RC4:n tiedossa olevien puutteiden vuoksi ratkaisua voi pitää väliaikaisena.

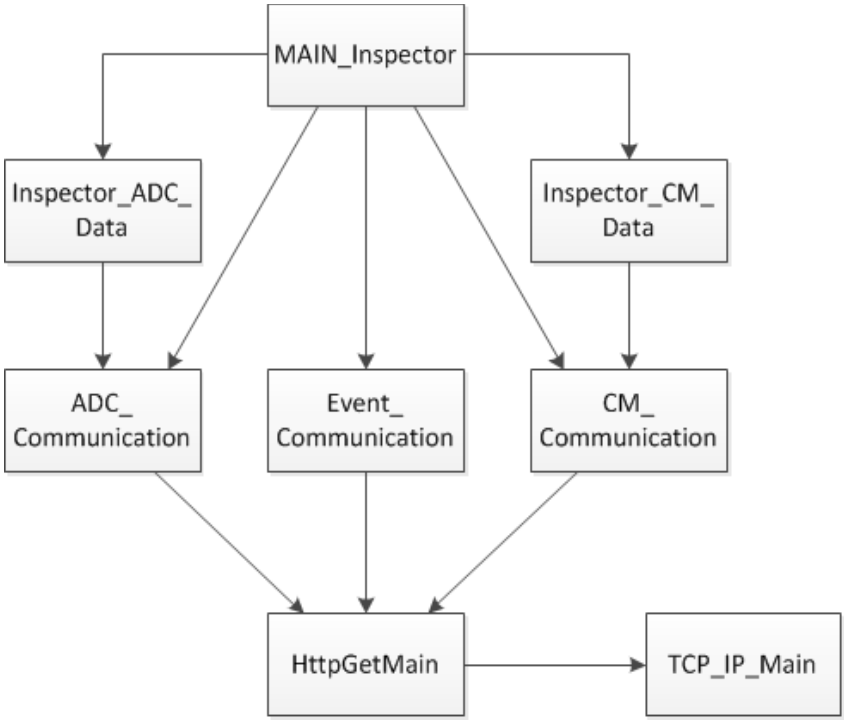
5.2 Arkkitehtuurin kuvaus

PLC-ohjelmoinnissa on tiettyjä erityispiirteitä muuhun ohjelmistokehitykseen verrattuna. TwinCAT2 ohjelmiston kehitysympäristö ei standardin IEC 61131-3 mukaisesti sisällä varsinaisia olio-ohjelmoinnin ominaisuuksia. Vaikka läheskään kaikki

olio-ohjelmoinnin tarjoamat mahdollisuudet eivät ole kyseisessä ympäristössä käytettävissä, on oliosuunnittelun periaatteita pyritty noudattamaan mahdollisimman paljon. Myös dynaamisen muistinhallinnan puuttuminen TwinCAT2 ympäristöstä vaikuttaa ohjelmiston toteutusyksityiskohtiin. Kehitysympäristön erityispiirteistä huolimatta ohjelmistokomponenteista on pyritty tekemään mahdollisimman helposti uudelleenkäytettäviä.

5.3 Tärkeimmät ohjelmistokomponentit

Tässä kappaleessa esitellään PLC-ohjelman tärkeimmät komponentit. PLC-ohjelman rakennetta ja tärkeimpien komponenttien suhdetta toisiinsa on havainnollistettu kuvan 13 kaaviossa.



Kuva 13: PLC-ohjelman rakenne

Seuraavaksi on käsitelty PLC-ohjelman tärkeimpiä komponentteja yleisellä tasolla. Komponenttien tarkat rajapintojen määrittelyt löytyvät kustakin koodilohkosta kommentteina aina kyseisen ohjelmakomponentin määrittelyn yhteydestä.

Main_Inspector

Main_Inspector on muista esitellyistä komponenteista poiketen tyypiltään ”Program” muiden ollessa toimilohkoja. Se toimii koko PLC-ohjelman pääohjelmana eli tämä

kyseinen ohjelmakutsu suoritetaan kokonaan jokaisella PLC-syklillä ja se kutsuu muita ohjelmassa tarvittavia komponentteja.

Inspector_ADC_Data
Inspector_ADC_Data sisältää Fadector-tuotteesta periytyneen toiminnallisuuden. Se seuraa logiikkakontrolleriin kytkettyjä tuloja, joihin on kytketty mitattavat signaalit, ja muodostaa niiden pohjalta ADC-datan. Tämän datan se tallettaa puskuriin palvelimelle lähettämistä varten.

Inspector_CM_Data
Tämä toimilohko käsittelee mittaustietoa ja analysoi siitä halutut tiedon CM-signaalien muodostamiseksi. Inspector_CM_Data pitää yllä mittaustietojen raakadatan puskurointia. Lisäksi se laskee tarvittavat ominaisuudet mittauksista ja määrittää lähetettävät tiedot omaan puskuriin. Laskenta suoritetaan annettujen parametrien mukaisesti, esimerkiksi on parametroitu kuinka pitkältä ajalta mittauksen keskiarvoa lasketaan.

CM_Communication
CM_Communication käsittelee Inspector_CM_Data:n tuottamaa lähetyspuskuria lähetettävistä tiedoista. Se jalostaa lähetettävät tiedot oikeaan muotoon eli JSON-viestiksi, joka tallennetaan http-kehykseen ja siirretään HttpGetMain-komponentille lähetettäväksi palvelimelle. CM_Communication myös valvoo viestin siirron onnistumisen ja poistaa tiedon lähetysrekisteristä, kun viesti saadaan siirrettyä onnistuneesti palvelimelle.

ADC_Communication
ADC_Communication käsittelee Inspector_ADC_Data:n tuottamaa tietorakennetta, johon ADC-signaalit on tallennettu. ADC-datasta se muodostaa JSON-standardin mukaista dataa, joka sisältää tiedot tilojen muutoksista aikaleimoineen. Kuten CM_Communication, myös ADC_Communication paketoi JSON-datan http-kehykseen ja kutsuu HttpGetMain toimilohkoa tiedon lähettämiseksi ja siirron onnistumisen seuraamiseksi.

Event_Communication
Event_Communication puolestaan huolehtii poikkeavista tilanteista kertovien tapahtumien kommunikoinnista palvelimelle muodostamalla niistä JSON-dataa http-kehykseen käärittynä. Tämän lisäksi Event_Communication valvoo ja käsittelee palvelimelta tulevat parametriviestit.

HttpGetMain

HttpGetMain valvoo palvelinyhteyden tilaa TCP_IP_Main-lohkoa käyttäen. Se käyttää TCP_IP_Mainin rajapintaa kontrolloidakseen yhteyden tilaa ja viestien lähetystä. Sen tehtäviin kuuluu myös huomata pakettien saapuminen ja välittää tieto eteenpäin tarvittavien toimenpiteiden suorittamiseksi.

TCP_IP_Main

TCP_IP_Main vastaa tietoliikenteestä palvelimelle ja se koostuu kolmesta osasta, TCP_IP_Socket_Connection, TCP_IP_Socket_Send ja TCP_IP_Socket_Receive. TCP_IP_Main luo TCP/IP-socketin TC_IP_Socket_Connection toimilohkoa kutsumalla. Samaa lohkoa kutsumalla socket voidaan sulkea ja vapauttaa, kun yhteyttä ei enää tarvita. TCP_IP_Socket_Receive:n avulla vastaanotetaan palvelimen lähettämä viesti. Tällöin valvotaan, että koko kehys on vastaanotettu onnistuneesti eikä dataa ole hukkunut matkalla. Vastaavasti TCP_IP_Send huolehtii sille annetun datan lähettämisestä palvelimelle. Yhteyden muodostukseen ja sulkemiseen, paketin lähettämiseen sekä paketin vastaanottamiseen nämä ohjelmalohkot käyttävät Beckhoffin valmista TCP/IP-kirjastoa.

Ohjelman muut komponentit

Yllä kuvattujen komponenttien lisäksi PLC-ohjelma pitää sisällään monia pienempiä komponentteja ja aliohjelmia, esimerkiksi merkkijonojen ja puskurirakenteiden käsittelyyn tarkoitettuja aliohjelmia. Lisäksi ohjelma käyttää Beckhoffin valmiita standardikirjastoja sekä TCP/IP-pinon toteuttavaa ohjelmakirjastoa.

6 OHJELMISTON TESTAUS

Toteutettua järjestelmää tullaan käyttämään käynnissä olevassa asiakasprojektissa, joten järjestelmän huolellinen testaus on tärkeää.

6.1 Testaussuunnitelma

Testaamista varten pystytettiin testiympäristö, joka mahdollisti CX8090 logiikan testaamisen käyttäen todellista palvelinta ja erilaisia kuormituksia. Tärkeimpiä testattavia ominaisuuksia olivat TCP/IP-protokollan toiminta ja luotettavuus sekä suorituskyvyn rajat ja mahdollisten pullonkaulojen tunnistaminen. Testaamisella pyrittiin myös saamaan tietoa mahdollisesti puskuroitavan tiedon kokoluokasta ja häiriöistä toipumisista. Testauksessa simuloitiin signaalin mittausta ja lähetystä kappaleessa 4.3 esiteltyjen toimintojen toimivuuden varmistamiseksi. Lisäksi testauksella varmistettiin ohjelman oikea toiminta erilaisissa poikkeustilanteissa, kuten yhteyskatkossa, virtakatkossa ja raja-arvojen ylittyessä.

6.1.1 Testilaitteisto

Testissä käytetyn PLC:n malli on Beckhoff CX8090. CX-sarjan logiikkakontrollerit ovat pieniä teollisuus-PC:ita, joissa on käytössä Windows CE käyttöjärjestelmä. Sen päällä ajetaan logiikkaohjausta puhtaasti ohjelmallisesti eli kyseessä on niin sanottu soft-PLC. PLC:n käyttämänä verkkoyhteytenä käytetään nykyaikaista ADSL-liittymää sekä erillistä 3G-modeemia. Tulevassa asiakasprojektissa tullaan mittaamaan tärinän voimakkuutta, josta pystytään päättelemään, onko laite käynnissä, ja tunnistamaan poikkeavan voimakas värähtely. Testeissä käytetään vastaavaa värähtelyn voimakkuutta mittavaa anturia, jota tullaan käyttämään myöhemmin asiakkaan prosessin mittaamiseen. Näin saadaan kokemusta myös anturin toiminnasta ja testitapauksia on helppo demonstroida anturia ravistamalla vaikka mitattavan suureen simulointikin on PLC-laitteistolla helppoa.

6.1.2 TCP/IP protokollan toiminta

Koska logiikka puskuroi lähetettävää dataa ja puskuria puretaan vasta onnistuneilla tiedonsiirroilla, ei yhteyden hetkellinen toimimattomuus saa aiheuttaa pysyviä ongelmia. Esimerkiksi yhteyden katkeaminen palvelimeen ei saa aiheuttaa lopullista toimimattomuutta, vaan järjestelmän täytyy toipua täysin automaattisesti yhteyden palauduttua. Vastaavissa laitteistoissa on aikaisemmin havaittu Windowsin TCP/IP-palvelun muistin roskaantuvan joissain tapauksissa, mikä aiheuttaa lopulta yhteyden jumittumisen. Vastaavien ongelmien välttämiseksi tullaan myös muistin kulutusta tarkkailemaan testissä.

6.1.3 Puskurointikyky ja muistin käyttö

Koska PLC puskuroi mittaustietoa omaan muistiinsa, muistia vapautuu vasta kun tiedot on onnistuneesti siirretty palvelimelle. Jos yhteys katkeaa pitkäksi aikaa, täyttyy muisti lopulta aiheuttaen mittaustietoon häviöitä. Testeissä on tarkoitus kartoittaa mitattavien tietojen (analoginen tai digitaalinen mittaus ja määrä) vaikutus hyväksyttävään yhteyden katkon pituuteen. Lisäksi tulee varmistaa järjestelmän automaattinen toipuminen yhteyden jälleen.

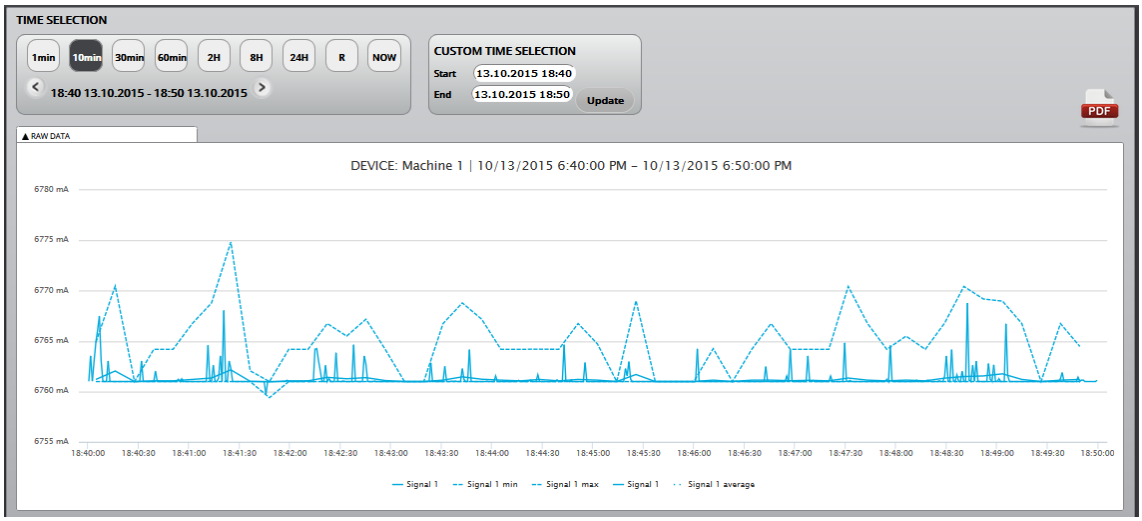
6.2 Testaamisen suorittaminen

Ohjelmistoa testatessa on simuloitu erilaisia testitapauksia PLC:lla ja verrattu tuloksia ohjelmiston vaatimusmäärittelyyn perustuviin odotettuihin tuloksiin. Toiminnot on testattu CX8090-kontrollerilla oikeaa palvelinta sekä verkkoyhteyttä käyttäen. Lisäksi järjestelmää on pidetty käynnissä pitkiä jaksoja luotettavuuden varmistamiseksi sovelluksen jatkuvassa käytössä. Muut tässä työssä kuvatut ominaisuudet on testattu toimiviksi oikean palvelimen kanssa, lukuun ottamatta RC4-salausta. Salauksen PLC-osio on testattu mustalaatikkotestauksena ilman palvelinta, jonne salausta ei ole vielä toteutettu. RC4 voidaan helposti ottaa käyttöön myös palvelimella käyttämällä tämän työn liitteenä 1 olevaa C#-kielellä tehtyä esimerkkitoteutusta. RC4-salauksen PLC-toteutuksen testauksessa on käytetty IETF:n (Internet Engineering Task Force) julkaisemia testivektoreita (IETF 2015).

6.3 Testien tulokset

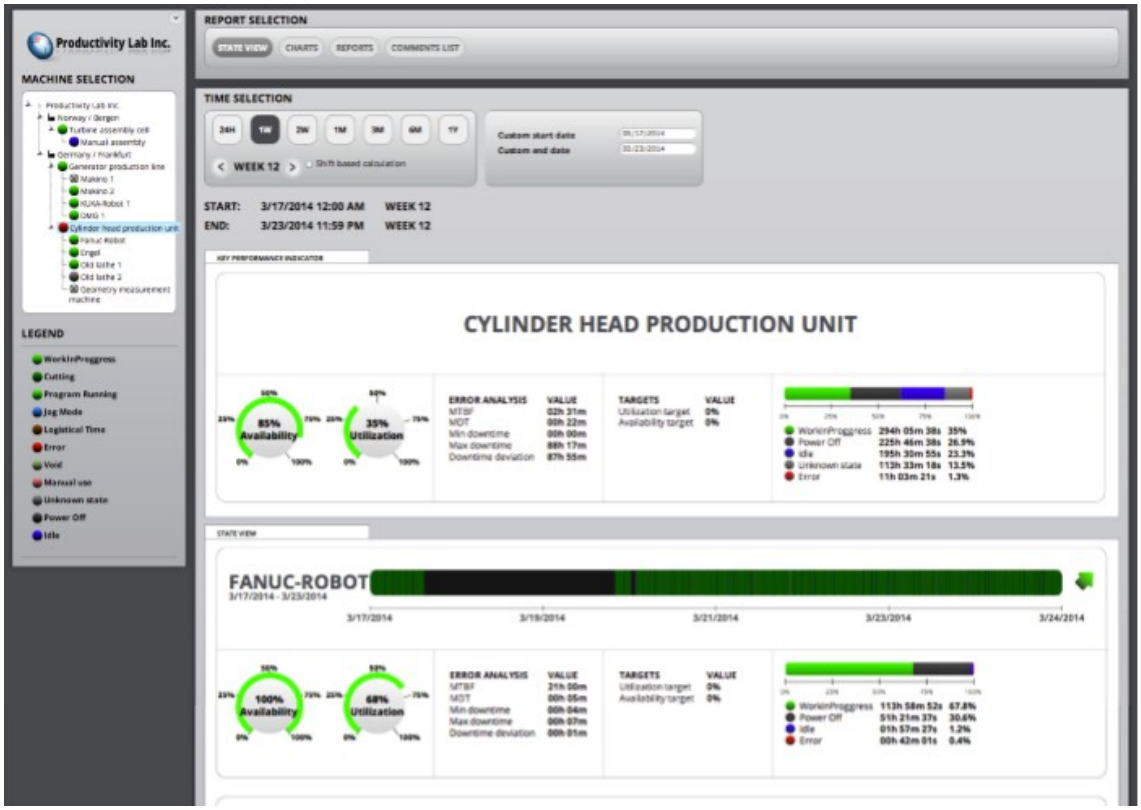
Ohjelmistoa testaamalla on varmistuttu sen oikeasta toiminnasta. Arkkitehtuuri on todettu toimivaksi kyseiseen sovellukseen ja PLC:n ja palvelimen toiminnassa ei ole havaittavissa merkittäviä ongelmia. Lisäksi järjestelmän toiminnan luotettavuus on osoittautunut hyväksi. Aikaisemmat huolet PLC:n muistin riittävyydestä ovat osoittautuneet laskennallisesti täysin hallittavissa oleviksi. Myöhemmin laskelmien tulokset on todettu oikeiksi myös käytännön testeissä. Lisäksi PLC selviytyy hyvin datan puskuroinnista poikkeustilanteissa, kuten yhteys- ja sähkökatkoista ilman että dataa pääsee häviämään. Myöskään aiempien kokemuksien perusteella pelättyjä Windowsin TCP/IP-palvelun muistivuotoja ei ilmennyt, joten ainakin nykyinen toteutus kyseisellä logiikkatyypillä todettiin luotettavaksi.

Signaalien esitys käyttöliittymässä on myös toimiva ja havainnollinen. Käytettävyyden vuoksi signaalin tarkasteltavaa jaksoa on helppo muutella ja kuvaaja skaalautuu automaattisesti käyttäjäystävälliseen muotoon. Esimerkki mitatusta signaalista käyttöliittymässä esitettynä on kuvassa 14.



Kuva 14: Inspector käyttöliittymä ja CM-signaali

Käyntitietoa mittaavan ADC-signaalin esitys käyttöliittymässä on esitetty kuvassa 15. Käyttöliittymässä on myös nähtävissä lasketut käyntiasteet, jonka lisäksi signaalista on saatavissa tarkempia tietoja käyttöliittymän avulla.



Kuva 15: Inspector käyttöliittymä ja ADC-signaali

Kokonaisuutena ohjelman tämänhetkiseen toimintaan voidaan olla tyytyväisiä. Järjestelmän perustoiminnallisuus toimii luotettavasti ja mahdollistaa ohjelmiston käyttämisen asiakasprojekteissa.

7 TULOSTEN TARKASTELU

Tämän työn päämääränä oli saada vastaukset kappaleessa 2 esitettyihin tutkimuskysymyksiin, jotka kohdistuivat käytettäviin tiedonsiirtoteknologioihin, kerättävän tiedon puskurointiin ja käyttökelpoisiin tietoturvaratkaisuihin.

Teknologiaselvitys
Tämän työn tuloksena on lisääntynyt ymmärrys teknologioista, joita voidaan käyttää automaattisesti kerätyn tiedon siirrossa PLC:lta pilvipalvelimelle. Käytettäviksi valittuja teknologioita, protokollia ja standardeja on esitelty. Käytössä olevat teknologiat ovat tällä hetkellä järjestelmän käytön mahdollistavia ja toimiviksi testattuja. Myös vaihtoehtoisia teknologioita on käyty läpi, ja erityisesti OPC UA mahdollistaisi vastaavan toiminnallisuuden tarjoten lisäksi hyödyllisiä lisäominaisuuksia tulevaisuuden tarpeisiin.

Tietoturvaselvitys
Työssä tarkasteltiin myös mahdollisesti järjestelmään sovellettavia tietoturvatekniikoita. Toistaiseksi PLC-ohjelmaan on implementoitu RC4-menetelmän mukainen salaus, joka voidaan ottaa käyttöön palvelussa kohtalaisen pienellä vaivalla. RC4 ei takaa parasta mahdollista tietoturvaa, mutta se on toistaiseksi riittävä tietoturvaratkaisu sovelluksen luonne huomioon ottaen. Tasokkaamman tietoturvan toteuttaminen itse ohjelmoiduilla mekanismeilla vaatisi huomattavasti lisää työtä, ja koska tämän työn suosituksena on siirtyä jatkossa käyttämään OPC UA standardia, eivät suuret ponnistelut uusien tietoturvaominaisuuksien kehittämiseksi ole perusteltuja ennen OPC UA standardin käytöstä tehtävää päätöstä. Tiedon siirto OPC UA-standardia käyttäen tulisi tarjoamaan vahvan sisäänrakennetun tietoturvan myös tulevaisuuden tarpeita varten. Jos siirtyminen OPC UA-standardin käyttöön päätetään hylätä, voidaan salauksen tasoa parantaa toteuttamalla AES-algoritmiin perustuva salaus tiedonsiirtoa varten.

Toteutettu ohjelmisto
Työn käytännön osuudessa kehitettiin järjestelmän PLC-ohjelmistoa. Kehitystä oli tehty paljon jo ennen kuin tämän työn toteutus pääsi kunnolla vauhtiin, joten merkittävimmät suunnitteluratkaisut oli jo päätetty. Lopputuloksena oli vaaditut ominaisuudet toteuttava ohjelma, joka on testaamalla osoitettu toimivaksi. Näin ollen uudet ominaisuudet voidaan ottaa käyttöön tulevissa projekteissa. Järjestelmän on todettu käytännössä suoriutuvan luotettavasti yhteys- ja virtakatkoista ja pystyvän puskuroimaan mittaustulokset luotettavasti.

7.1 Ohjelmiston jatkokehitys

Ohjelmiston jatkuva ylläpito ja jatkokehitys on välttämätöntä ohjelmistotuotteeseen perustuvalle liiketoiminnalle. PLC on osoittautunut hyväksi tavaksi kerätä monipuolisesti erilaista tietoa moninaisista teollisuuden prosesseista ja tapahtumista. Myös käyttöliittymän kehityksessä on saadun palautteen perusteella onnistuttu huomioiden hyvin samalla loppukäyttäjän todelliset tarpeet. Suuri osa Inspector tuotteen jatkokehityssuunnitelmista koskee käyttöliittymää ja esimerkiksi eri mobiililaitteille toteutettuja käyttöliittymiä, joita ei kuitenkaan tässä työssä käsitellä tarkemmin.

OPC UA

Tämän työn merkittävin jatkokehityssuositus on toteuttaa OPC UA standardin mukainen tiedonsiirto PLC:n ja palvelimen välille. Merkittävä etu OPC UA:n käyttöönotossa on sen sisäänrakennettu vahva tietoturva sekä binäärimuotoinen tehokas tiedonsiirto. OPC UA:aan löytyy valmiita ohjelmistokomponentteja, joten muutoksen toteutus onnistunee kohtuullisella vaivalla palvelimelle. Myös Beckhoffilta löytyy valmiita komponentteja, joilla OPC UA palvelin voidaan toteuttaa ja konfiguroida CX-sarjan logiikkakontrollerille. Lisäksi OPC UA vaikuttaa lupaavasti yleistyvän teollisuudessa, joten sen uskotaan olevan tulevaisuudessa varsin yleisesti käytetty standardi. Kun palvelimelle toteutetaan OPC UA tuki, se tulee mahdollistamaan tulevaisuudessa laitteiden suoraviivaisen ja helpon liittämisen Inspector palveluun, jopa ilman PLC:n tarvetta tiedon välittäjänä.

Kasvaviin datamääriin varautuminen

Tällä hetkellä ongelmatilanteissa puskuroidun tiedon määrää rajoittaa PLC:n virtakatkon yli säilyvän muistin määrä, jonka kasvattaminen ei ole mahdollista toistaiseksi ilman huomattavaa laitekustannuksien kasvua. Ongelma voidaan kuitenkin kiertää kehittämällä mekanismi, jolla pitkän yhteyskatkon sattuessa tallennetaan puskuroitua tietoa tiedostoon PLC:n käyttämälle muistikortille, jota CX8090-kontrolleri tukee ainakin 4GB asti (Beckhoff 2015). Yhteyden palauduttua PLC lukisi tiedostot ja lähettäisi tiedot palvelimelle uusien mittauksien ohella. Näin puskuroitavan tiedon määrää olisi mahdollista kasvattaa merkittävästi.

8 YHTEENVETO

Teollisuuden tuotantojärjestelmistä automaattisesti kerättävä tieto tarjoaa suuria mahdollisuuksia tuotantotehokkuuden parantamiseen. Saatavan tiedon määrä kasvaa nopeasti ja järjestelmät integroituvat yhä tiiviimmin toisiinsa. Lisäksi tuotantojärjestelmien älyn lisääntyminen tulee ohjaamaan teollisuuden toimintamalleja. Suuret tietomäärät eli niin sanottu big data tulevat tarjoamaan merkittäviä mahdollisuuksia prosessien kehittämiseen ja kokonaan uuden liiketoiminnan synnyttämiseen. Lisäksi nykyaikaisissa automaatiojärjestelmissä ohjelmistojen merkitys korostuu hallittavien tietomäärien ja integraation kasvaessa. Erilaisilla suunnittelumalleilla pyritään ottamaan huomioon automaation erikoispiirteet ohjelmistosuunnittelussa. Myös yleiset ohjelmistotuotannon periaatteet, kuten käytettävyys ja ylläpidettävyys on huomioitava automaatiojärjestelmien ohjelmistoissa.

Tiedonsiirtoon on käytettävissä useita menetelmiä ja standardeja. TCP/IP-protokollaperhe on tiedonsiirrossa hyvin yleisesti käytetty ja HTTP mahdollistaa sovelluksien kehittämisen käyttäen vakiokomponentteja, kuten tässä työssä käsitelty sovellus osoittaa. Myös tietoturvan toteutukseen on tarjolla runsaasti erilaisia menetelmiä. Tässä työssä käsitellyt erilaiset tekniset tietoturvaratkaisut toteuttaisivat tarvitut ominaisuudet, mutta valmiita komponentteja PLC-ympäristöön on valitettavan niukasti saatavilla.

Tulevaisuudessa tehokas tapa toteuttaa tiedonsiirto olisi OPC UA-standardi. OPC UA vaikuttaa yleistyvän teollisessa käytössä. Standardi pitää sisällään myös vahvat tietoturvaominaisuudet, joten kyseinen standardi soveltuisi hyvin PLC:n ja pilvipalvelun väliseen tiedon siirtoon.

LÄHDELUETTELO

Beckhoff [WWW], Viitattu 31.10.2015. Luettavissa https://www.beckhoff.com/

Comer D. E., 2000, Internetworking with TCP/IP Vol. 1: Principles, Protocols, and Architecture, 4th ed., Pretince Hall, Upper Saddle River, New Jersey, 366 s.

Douglass, B., 2009, Real-time agility : the harmony/ESW method for real-time and embedded systems development, Addison-Wesley, Upper Saddle River, New Jersey, 522 s.

Evans, P. C., Annunziata, M., 2012, Industrial Internet: Pushing the Boundaries of Minds and Machines, General Electric, 332 s.

Fall K. R., Stevens W. R., 2011, TCP/IP Illustrated, Volume 1, The Protocols, 2nd edition. Pearson education inc. USA, 1056 s.

Haikala, I., Märijärvi, J., 2004, Ohjelmistotuotanto, Talentum Media Oy, 440 s.

IETF [WWW], Viitattu 31.10.2015. Luettavissa https://tools.ietf.org/html/rfc6229/

InSolution [WWW], Viitattu 26.9.2015. Luettavissa http://www.insolution.fi/

JSON standardi [WWW], Viitattu 7.9.2015, Luettavissa www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Juran, J., 1998, Juran's quality handbook, McGraw-Hill, 1872 s.

Jännes, J., 2011, Käyttövarmuuden ja turvallisuuden hallinta suunnittelun alkuvaiheissa, Diplomityö, TTY, 108 s.

Järvinen, P., 2003, Salausmenetelmät, Docendo Finland Oy, 385 s.

Kagermann, H., Wahlster, W., Helbig, J., 2013, Recommendations for implementing the strategic initiative INDUSTRIE 4.0. Final Report of the Industrie 4.0 Working Group. Linda Treugut, M.A., Acatech – National Academy of Science and Engineering, 80 s.

Kerttula, E., 1999, Tietoverkkojen tietoturva, Oy Edita Ab, 510 s.

Kuikka, S., 2007, Kurssin ACI-32020 Automaation reaaliaikajärjestelmät luentokalvot, TTY.

Laihonen, H., Hannula, M., Helander, N., Ilvonen, I., Jussila, J., Kukko, M., Kärkkäinen, H., Lönnqvist, A., Myllärniemi, J., Pekkola, S., Virtanen, P., Vuori, V., Yliniemi, T., 2013, Tietojohtaminen, Tampereen teknillinen yliopisto, 84 s.

Leclin, O., 2006, Laatu yrityksen menestystekijänä, Talentum, 408 s.

Mahnke, W., Leitner, S., H., Damm, M., 2009, OPC Unified Architecture, Springer, 339 s.

MTConnect Institute, MTConnect Standard, Part 1 – Overview and Protocol, version 1.3.0, Viitattu 19.9.2015, Luettavissa www.mtconnect.org

NIST (National Institute of Standards and Technology), 2011, The NIST Definition of Cloud Computing, Computer Security Division, Information Technology Laboratory, Gaithersburg, 7 s.

OPC Foundation [WWW], Viitattu 4.10.2015, Luettavissa https://opcfoundation.org/

Renjie, H., Feng, L., Dongbo, P., 2010, Research on OPC UA security, 2010 5th IEEE Conference on Industrial Electronics and Applications

RFC7231, HTTP/1.1 Semantics and Content [WWW]. Viitattu 26.9.2015, Luettavissa https://httpwg.github.io/

RFC5246, The Transport Layer Security (TLS) Protocol, Version 1.2. [WWW]. Viitattu 7.10.2015, Luettavissa https://tools.ietf.org/html/rfc5246

RFC7540, HTTP/2 [WWW]. Viitattu 26.9.2015, Luettavissa https://httpwg.github.io/

Ruohotie, J., 2006, PHP-pohjaisen nopean kehityksen sovelluskehyksen suunnittelu ja toteutus, Diplomityö, TTY, 61 s.

Scholten, B., 2007, The road to integration; a guide to applying the ISA-95 standard in manufacturing., ISA – Instrumentation, Systems, and Automation Society, 241s.

Sydänmaanlakka, P., 2007, Älykäs organisaatio, Talentum Media Oy, 299 s.

Tanenbaum A. S., van Steen M., 2007, Distributed systems: principles and paradigms, 2nd ed., Pretince Hall, Upper Saddle River, New Jersey, 686 s.

Välisalo T. & Rouhiainen V., 2000, Luotettavuusjohtaminen työkoneteollisuudessa. VTT Tiedotteita 2061. Valtion teknillinen tutkimuskeskus (VTT), Espoo, 58 s.

LIITE 1: RC4-ALGORITMIN TOTEUTUS C#-KIELELLÄ

```
public void RC4(ref Byte[] bytes, Byte[] key)
{
    Byte[] s = new Byte[256];   //s-box
    Byte[] k = new Byte[256];   //key
    Byte temp;                  //substitution buffer
    int i, j;                   //counters

    //S-Box & Key (KSA)
    //1 Iteration for each S-Box & key element
    for (i = 0; i < 256; i++)
    {
        s[i] = (Byte)i;                     //Set Values to 0...255
        k[i] = key[i % key.GetLength(0)];   //Fill k with key, append
                                            //if neccesary
    }

    j = 0;
    //1 Iteration for each S-Box element
    for (i = 0; i < 256; i++)
    {
        //Calculate j depending on the key
        j = (j + s[i] + k[i]) % 256;
        //Swap s[i] and s[j]
        temp = s[i];
        s[i] = s[j];
        s[j] = temp;
    }

    //Pseudo Random Generating Algorithm (PRGA)
    i = j = 0;
    //1 Iteration for each byte of the plaintext
    for (int x = 0; x < bytes.GetLength(0); x++)
    {
        i = (i + 1) % 256;       //Count-up i 0...255 to match S-Box
                                 //index
        j = (j + s[i]) % 256;    //Calculate j depending on S-Box,
                                 //match index
        temp = s[i];             //Swap s[i] and s[j]
        s[i] = s[j];
        s[j] = temp;
        int t = (s[i] + s[j]) % 256;      //s[i]+s[j] as index for an
                                          //S-Box element
        bytes[x] ^= s[t];                 //XOR plaintext-byte with S-
                                          //Box element
    }
}
```